NOVA IMS Information Management School

MGI

Mestrado em Gestão de Informação
Master Program in Information Management

Extract, Transform, and Load data from Legacy Systems to Azure Cloud

Ioudom Foubi Jephte

Internship Report presented as the partial requirement for obtaining a Master's degree in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Extract, Transform, and Load data from Legacy Systems to Azure Cloud

by

Ioudom Foubi Jephte

Internship report presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Knowledge Management and Business Intelligence

Advisor*: professor* Doutor Flávio Luis Portas Pinheiro

February 2021

Acknowledgments

To Accenture Portugal, for seeing the potential in me and accepting me to join their consulting team; More so, the warm welcome from consultants and the constant support during the internship.

To my supervisor, Professor Doutor Flávio Luis Portas Pinheiro, who provided answers to my questions, I am grateful. Thank you for the fantastic class experience and for inspiring me to undertake this path in data engineering.

To my father Foubi, my mother Nguemdjo, my brothers Nwatchou and Pouomegne, thank you for the enormous sacrifice, constant support, and motivation whenever I need them. You are all a blessing to me.

To Mr Olufemi, thank you for the help in my journey to come to Portugal and for your mentorship. To Mrs Singum, thank you for believing in me and supporting me in making my dream possible. To Zommi, thank you for your unconditional support whenever I needed it and your amazing heart. Thank you for your true friendship and supportive words, the MAJAG family. To Felipe, thank you for your professional advice and support. To God, almighty, for guiding me every day.

Finally, I say thanks to all those who helped me in one way or the other during this journey.

Abstract

In a world with continuously evolving technologies and hardened competitive markets, organisations need to continually be on guard to grasp cutting edge technology and tools that will help them to surpass any competition that arises. Modern data platforms that incorporate cloud technologies, support organisations to strive and get ahead of their competitors by providing solutions that help them capture and optimally use untapped data, and scalable storages to adapt to ever-growing data quantities. Also, adopt data processing and visualisation tools that help to improve the decision-making process.

With many cloud providers available in the market, from small players to major technology corporations, this offers much flexibility to organisations to choose the best cloud technology that will align with their use cases and overall products and services strategy. This internship came up at the time when one of Accenture's significant client in the financial industry decided to migrate from legacy systems to a cloud-based data infrastructure that is Microsoft Azure cloud.

During this internship, development of the data lake, which is a core part of the MDP, was done to understand better the type of challenges that can be faced when migrating data from on-premise legacy systems to a cloud-based infrastructure.

Also, provided in this work, are the main recommendations and guidelines when it comes to performing a large scale data migration.

Keywords

Modern data platform; data lake; delta lake; data integration; data transformation; data loading

Index

List of Figures

List of Tables

List of Abbreviations and Acronyms

ETL	Extraction, Transformation, and Loading
EDP	Enterprise Data Platform
MDP	Modern Data Platform
R&D	Research & Development
ERP	Enterprise Resource Planning
HRM	Human Capital Management
GDPR	General Data Protection Regulation
POSIX	Portable Operating System Interface
AWS	Amazon Web Services
IaaS	Infrastructure as a Service
GCP	Google Cloud Platform
MLlib	Machine Learning Library
S3	Simple Storage Service
SLA	Service Level Agreement
ACID	Atomicity, Consistency, Isolation, Durability
DB	Database
HDFS	Hadoop Distributed File System
CDC	Change Data Capture
IBM	International Business Machines
NDA	Non-Disclosure Agreement
AI	Artificial Intelligence
SQL	Structured Query Language
NoSQL	Not Only Structured Query Language
OLTP	On-Line Transaction Processing
OLAP	On-line Analytical Processing
RDD	Resilient Distributed Dataset
CPU	Central Processing Unit
RDBMS	Relational Database Management System

1. Introduction

The 21st century saw the exponential growth of data generated with the development of new technologies necessary to deal with the challenge of storing and processing the increasing streams of data, a challenge that led to the popularization of the concept of Big Data. Big Data can be used to describe a range of different ideas, from collection and processing of large amounts of data to the advanced computational techniques often necessary to extract analytical value from the collected data [46]. However, there is no clear definition of the term big data, and researchers from different academic backgrounds often take different definitions for what it means.

IBM, a global technology company, proposed the four dimensions of big data usually called the V's of Big Data to ease the understanding of the phenomenon of big data. These V's include volume, velocity, variety and veracity. Volume refers to the massive amounts of data generated every second which can include videos, photos, emails and sensor data produced and shared every second, which increased to Exabyte, zettabyte and brontobyte of data in terms of the total size. Velocity can be seen as the speed at which data is generated and the speed at which these data moves around, for example, the milliseconds taken by big data algorithmic trading systems to analyze stock market data and assisting traders to make decisions to buy and sell financial securities on the stock market. Variety refers to the different types of data available for use and veracity refers to the level of trustworthiness of the data [1].

Traditional data storage solutions, such as relational databases adopted years ago, residing on old on-premise legacy systems, cannot handle the complexity and the scalability needs that come with holding the variety of data types produced today in a single centralised location. It poses new dilemmas that organisations need to face in order to meet their business needs and regulatory policy constraints.

In order to facilitate the storing of massive data sets, while minimizing the internal burden of having multiple teams or departments accessing data for different needs, organisations embraced new storage solutions commonly referred to as Data Lakes. With Data Lakes, storage is scalable according to the needs of an organization. It can reach volumes in the order of petabytes while allowing the storage of both structured and unstructured data formats. These overcome many of the limitations set by traditional storage solutions.

Organisations adopting data lake solutions empower themselves to exploit better, explore, and analyse data to improve their value proposition towards clients. In that context, this report explores a project that was developed within the Applied Intelligence team of Accenture Portugal. The Applied Intelligence team focuses on helping Accenture clients to build a modern data platform(MDP) that leverages the cloud,

distributed system paradigms, big data technologies and best practices to improve their processes, products, and services.

1.1. Company overview

Accenture is a global professional services company, listed among the Fortune 500 Global companies and incorporated in Dublin (Ireland) since 2009, as of 2019, Accenture had more than 492,000 employees serving clients in more than 120 countries [2]. Ninety-one of the Fortune Global 100 are Accenture's current clients, and as for global fortune 500, more than three-quarters are also their clients [2].

Some of the industries Accenture has clients in include: Aerospace and Defence, Industrial Equipment, Insurance, Life Sciences, Automotive, Banking and Capital Markets, Chemicals, Communications, Consumer Goods and Services, Natural Resources, Public Service, Retail, Software, Energy, Health, Travel, and Utilities.

Accenture has four primary service areas including:

i. The strategy and consulting service arm of Accenture [3], helps organisations to design and implement programmes with technology at the centre. This arm supports organisations in planning and implementing their technology-driven transformations to improve all their business units. Accenture formulates strategies and identifies potential technological components such as AI, automation, data platforms, and digital marketing that will enable the rapid adoption and implementation of these strategies to make the organisation more competitive in its specific industry and generate wealth.

ii. The interactive service arm of Accenture [4], helps businesses become more agile by responding to evolving business landscapes, higher employee and customer expectations, often sparked by digital technologies. Also, it supports businesses to maximise the value from new routes to market, deploying advanced analytics, enhancing digital trust with robust security and enhancing digital engagement. Some past work of this arm includes the design and implementation of a personalised customer experience system for carnival corporation cruise making use of IoT and stream analytics to enable the company to offer customised services to each of their customers while on board of ships.

iii. The technology service arm of Accenture [5], provides core technical services, in-house and partner products to Accenture clients. Through the Accenture labs, this arm incubates new concepts and applies the latest technologies to deliver breakthrough solutions for businesses and society. The lab also has specialised R&D groups who investigate and use new technologies to help organisations provide breakthrough solutions to their current and future challenges. This

arm offers application services to organisations to help them reinvent their enterprise application portfolio, from the development of new applications leveraging on the cloud and blockchain, to the modernisation, management, and maintenance of existing applications. Accenture covers all stages of the application lifecycle.

iv. The operations service arm of Accenture [6], offers Business Processing Outsourcing (BPO) services. Accenture operations through its human-machine operating engine – SynOps, combines the best people, processes, and technology to help clients reshape their organisations with a new and more connected operating model. Through this arm, Accenture works with some of the biggest social media networks in the world to provide digital content review services to help eradicate content that violates user policies or terms of services.

1.2. The team and Activities

During the internship at Accenture Portugal, the Applied Intelligence team handled the MDP project. The capabilities offered by this team include artificial intelligence, intelligent automation, data and analytics, enterprise data management, search, and content analytics.

The Applied Intelligence team harnesses the power of cutting-edge technologies to help businesses shape a clear vision and develop new data supply chains by instilling greater trust in data and exploring improved ways to manage it. The applied intelligence team also helps clients to modernise their data platforms by supporting the building of data architectures to capture, curate, and store the correct data using a combination of cutting edge technologies.

Additionally, the applied intelligence team also offers the Intelligent Data Suite (IDS) [7], which helps businesses go beyond data silos. This suite helps clients discover and access data anywhere in their enterprise, then classify data by industry context, qualify data by verifying its trustworthiness and validity, and finally consolidate and prepare the data for analytics while comparing with industry standards. This team has strong alliances with technology providers including Microsoft, Amazon Web Services, Google Cloud, Informatica, IBM, SAP, Tableau, Adobe, Oracle, Salesforce, SAS, Qlik, Cloudera, and Snowflake [31].

The applied intelligence team did the MDP project for a major banking financial institution in Portugal. This project aimed to help the client build a new data platform that was going to leverage the Azure cloud, Databricks delta architecture, and other big data technologies to facilitate enterprise processes, operations, decision making, product, and services development.

The member's responsibilities within the team and project included:

- Gathering business requirements from the banking financial institution related to the MDP to understand the business needs, then translate these requirements into technical requirements and build a work plan to implement them.
- Based on the technical requirements, build different zones of the organisation's delta and data lake on Azure data lake Gen 2.
- Develop streaming ETL scripts on databricks to extract and process data from the client's legacy system and load on data lake Gen 2 zones and Azure Cosmos DB.
- Refactor all the codebase of the entire MDP project, build Unit-test on databricks to test all the code and build pyTests to automate testing of code.
- Research new workflows about databricks and other cloud services used in the project to ease the work of the team and solve technical issues faced within the project.

1.3. Internship Goals

With the increase in internet usage, customer needs have changed over time, in particular users among the middle and younger age groups who increasingly request more digitally inclined services. Such a shift towards digital products reflects in the popularity of digital banks such as N26, Revolut, Bunq.

Traditional banking and financial institutions with little to no digital services face challenges to adapt to these new customers' needs and trends. To develop competitive digital products and services, these financial institutions have to change or improve their technology and data infrastructure to enable the development of new digital services in an agile way.

This internship took place in the applied intelligence team of Accenture Portugal at the time when one of their major clients was building and deploying a modern data platform, making use of Azure cloud and big data technologies. This new platform was going to enable this client to develop more digital products and services, support real-time stream analytics, and at the same time, let engineers monitor in real-time any activity on the MDP.

The main intent of the internship was to participate actively in the development of modules to efficiently and effectively proceed with the extraction, transformation, and loading of data from legacy systems to the Azure cloud.

2. Theoretical Framework

This section provides a theoretical overview of the most relevant frameworks, tools, and concepts related to traditional and modern data infrastructures, distributed computing systems, and big data necessary for the understanding of the project reported in this document.

2.1 Data Platform as a Data Storage Solution

There are multiple definitions of data platforms. Looker, a leading global data visualization and business intelligence organizations owned by Google, defines a data platform as an integrated technology solution that enables the governance, access and delivery to users, data applications and other application of data located in databases for strategic business purposes [8]. Splunk Technology defined a data platform as a complete solution for ingesting, processing, analyzing and presenting the data generated by the systems, processes and infrastructures of the modern digital organization [9].

A data platform can serve strategically modern organizations in the following ways:

- **Availability**: A data platform that is connected directly to a database will ensure data teams within the organization can have access to the right data at the right time, without delays in processing large volumes of data or data request.

- **Security**: data platforms can help enforce authentication across all storages or data, providing access to data only to the authorized individuals or groups. Also, authentication tools on data platforms help to track those accessing data and keeping metadata or logs so that in the long run if there is an issue, it can be used to trace the cause(s).

- **Governance**: A data platform can enable businesses to manage their data governance strategy better, including what data is collected, who can access it, and when data has expired according to the data protection and privacy regulations.

- **Delivery**: A data platform can include scheduling reports, dashboards and proactive alerts for predetermined conditions. This functionality helps eliminate bottlenecks and deliver reliable and accurate data to authorized users at the moment of relevance. It also enables the use of APIs to deliver data to other tools for specialized or advanced forms of analysis, such as data science and AI/Machine learning workflows.

- **Centralization**: Through a data platform, organizational silos can be dissolved to enhance collaboration and effective decision; this is achieved when the data platform combines data from different sources into one single database or location [8].

2.2 Traditional Data Storage Solutions

Traditional data storage solutions here, refers to traditional databases, which do not make use of the distributed system and big data paradigms or concepts within their design. In the early days of data storage within organizations, data was organized within files. However, storing data within separate files adds unnecessary burdens and setbacks to organizations [10], which include, for instance:

- **Redundant data**: Storing the data required by each application program resulted in duplicated data. Redundant data is dangerous because it occupies extra storage space and causes data inconsistency.
- **Isolated data**: Data stored in separate files is very difficult to process altogether. Processing challenges increase when the data files are of different formats.
- **Program-data dependence**: The physical structures of files and data records are defined in the program code, which will make it difficult to change existing structures. However, if data structure changes are necessary, all programs that have access to the changed file must be modified.
- **No Ad-hoc queries**: To obtain new reports, new programs must be developed to generate these reports, as there are no possibilities for users to ask ad hoc queries.

All the setbacks that are introduced by organizing and storing data into files, as described above, are caused by two main factors: The data definitions are embedded in the application programs instead of being stored together with the data, and there is no data access point and processing capabilities other than those provided by the application programs.

To overcome and address the above-mentioned limitations, alternative storage solutions were developed and materialized in the form of databases based on a network or hierarchical data models. These databases still failed to provide an adequate solution to all the storage needs of organizations. In particular, running queries (that is, extracting data from these systems) was very complex because they were navigationally oriented and also due to the lack of sufficient data independence. [10]

In the 1970s a new form of databases was introduced by Edgar Frank Codd. Following this innovation from Frank Codd, by the late 1970s and 1980s, database management systems specific to handle these databases were introduced and become known as Relational Database Management System (RDBMS).

A relational database management system is a computer program that enables anyone to create, maintain, and manage a relational database. Nowadays, RDBMS are particularly well-known for their standard query language – Structured Query Language or SQL—that offers a simple and easy to learn interaction layer for users to access and manipulate data stored in databases.

Within organizations there are traditionally two storage solutions/roles implemented on top of RDBMS; OLTP also considered as operational database/system and OLAP also considered as a data warehouse.

2.3 Operational database and Data Warehouse

A database is said to be operational if used to store and manage daily transactions of organizations. In that sense, these databases are associated with OLTP (On-Line Transaction Processing) that are designed to operate as a real-time system that tracks daily operations or transactions that are small but can arrive in massive numbers, for example, in-store purchases, booking reservations, and order entry. Within OLTP, one key aspect is the atomicity of database transactions. Atomicity indicates whether a transaction succeeds entirely or fails entirely, thus, cannot be in an intermediate state.

Using OLTP for analytical processing can raise challenges as data models are not adapted for efficient analytical activities. In contrast, On-line Analytical Processing (OLAP) systems have been the main paradigm to set up and design Data Warehouses, databases built for a purely analytical purpose. Data warehouses have different definitions, for example, Turban [11] defines a data warehouse (DW) as a repository of current and historical data that is used to support decision making and structured in a way that it can be used for analytical processing activities. Other approaches proposed by Inmon [12] to describe a data warehouse makes use of its characteristics:

- **Subject Oriented**: Data is organized by subjects or themes such as sales, manufacturing, marketing, etc. By providing subject-oriented data, the DW allows its users to have a more comprehensive view of the organization and not just an operational view. [12]
- **Integrated**: Data in a data warehouse is usually collected from different sources to be placed together into a consistent format. To do so, naming conflicts, encoding structure, etc., need to be dealt with beforehand so similar data is scaled in the same way. The outcome is a totally integrated data warehouse. [12]
- **Time-Variant**: Data that resides in a data warehouse deliver information from the historical perspective and aren't mainly needed to provide current states. Historical data present in the data warehouse allows trend or deviation detection along with forecasting, all depending on the business context. All data warehouses need to have a time dimension. [12]
- **Non-Volatile**: Once the data are entered into the DW, they can't be altered. Changes are recorded as new data. Meaning that data is read-only and updated at set intervals depending on the organization's needs. [12]

Data warehouse models

Within data warehouses, multiple models and architectures have been proposed for storing information in a more efficient manner or to provide a better vertical synergy between the different layers of data storage in an organization. The two most popular data warehouse models/architectures are the Inmon and Kimball model.

i. Inmon Model

The Inmon model is considered as the enterprise data warehouse (EDW) and follows a Top-down design approach. The EDW and the top-down approach refers to the implementation of a data warehouse that covers the whole organization which is used to serve underlying departments within the organization. Within the Inmon model, all data marts are derived from the EDW to be used by individual departments independently from other departments.

ii. Kimball Model

The Kimball model focuses on designing a data warehouse around the idea of data-marts and is considered a Bottom-up approach. With this approach, data marts are created for individual departments within the organization and later merged to get a large data warehouse covering the organization. A data mart can be considered as a simple type of data warehouse that focuses on one subject or functional area for example sales, account management, marketing. There are three types of data marts namely, dependent, independent and hybrid data marts. Dependent data marts enable organizations to combine their data into one data warehouse. Independent data marts are created without making use of the central data warehouse within an organization and Hybrid Data marts enable the integration of data from different sources other than the data warehouse.

Data warehouse schema

A schema is a logical representation of an entire database, showcasing fact and dimension tables within the database. A fact table is a table that contains measures for example sales amounts totals, quantities and prices, while a dimension table, is a table that contains attributes describing the data in a fact table, for example, sales country locations and dates. There are four popular schema types in data warehouses including:

- **Star schema**

The star schema contains one or more dimension and fact tables. The entity-relationship between fact and dimension tables forms a star shape in which a fact table is connected to many dimension tables. In

this type of schema, dimensions only have one level. Data in this schema is denormalised which makes queries to run faster but at the same time causes the presence of redundant data.

- **Snowflake schema**

The snowflake is an extension of the star schema where dimensions have more than one level. Because the data in snowflake schema is normalized, the data is split into additional tables to avoid data redundancy. Processing of data with this type of schema might be slow due to complex joins needed to retrieve data.

- **Galaxy or Fact Constellation schema**

The galaxy schema has multiple fact tables that share dimension tables between each other. This schema is called galaxy because its shape forms a collection of stars. Shared dimensions are called conformed dimensions. Usually, this schema is built by splitting a star schema into multiple star schemas.

Challenges associated to Traditional Data Storage Solutions

The challenges associated with traditional data warehouses propounded by [13] are explored in the next paragraphs.

I. **Rigid structure**: One of the most recurrent loop-hole of traditional data warehouses is their lack of data modelling flexibility. In today's ever-changing business environment, with the constant development of new applications, high consumption of technology products and services, organisations must-have information on demand to respond faster to any change and adapt to market changes or shifts. With rigid traditional data warehouses, any change in the data model might take several months to go through an approval and the intervention of technical experts, which is arguably an incompatible setting in light of the above-mentioned pressures.

II. **Complex architecture**: To cope with constantly evolving requirements, organisations buy different technologies that lead to a more complex architecture with numerous data silos that generate new challenges. These challenges include the lack of integration caused by the unavailability of native integration across standard processes due to multiple technologies in the complicated infrastructure, which often leads to data governance problems and absence of agility. Additionally, complex infrastructures make it difficult to access a single source of truth, that leads to challenges in generating actionable insights. Besides, most of these tools have similar capabilities. Hence they appear as duplicate technologies that will not bring anything new to the infrastructure.

III. **Slow performance**: with the rapid rise in data volumes generated by organisations today, the performance of their platforms can be affected and cause delays in prompt reporting. Data preparation and consumption latency can portray the loopholes of a data warehouse. Unprecedented quantities of data can explain latency during data preparation, and that can halt traditional data warehouses. Additionally, the large volumes of data cause data migration challenges in traditional data warehouses. Also, duplicated but unused data in these traditional data warehouses hinder data preparation procedures. There is a hindrance of data consumption by longer running times of queries caused by increasing data quantities, data sets and complex analytical requirements.

IV. **Outdated technology**: Traditional data warehouses are built on rigid platforms and with little or no update possibilities. With growing competition and the high need for on-time decision making, these data warehouses can turn into barriers due to their inability to store and process vast volumes of data with different data types. Also, due to the massive cost associated with upgrading the hardware on which they run, for example purchasing and installing expensive CPU's and data centre level equipment.

V. **Lack of data governance**: Moreover, data governance can be a big issue with traditional data warehouses. The Data Governance Institute defines data governance as the practice of decision making and authority for information-related matters [14]. Creating a bond between people, processes, and technology leads to better data governance. [13] Concerning data governance, traditional data warehouses can disrupt data delivery value chain in the following ways:

- When looking at source systems, in case of organisations wanting to change them, traditional data warehouses can complicate impact analysis. In some instances, they might make it challenging to map and catalogue new systems without damaging data governance rules.

- Within traditional data warehouses, ETL processes might fail to produce standard log details, which makes reviewing and querying them difficult. Also, there might be a lack of consistent methods, tools and controls to ensure the correct processing of sensitive data in ETL processes. Note that loading data quickly into a traditional data warehouse can hinder data governance structures. Traditional data warehouses often lack standardised data models and extra metadata needed to facilitate semantics-based discovery and they might not support data segmentation based on standard rules.

2.4 Big Data

In the past, most databases and applications were located on a single machine or framework with all their operations being performed on that single machine. With the growth of data and the need to make more complex operations, this infrastructure started showing limitations such as the lack of scalability capabilities.

Scalability (Scale-Up and Scale-Out)

According to Gartner [15], Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands. With the constant growth of data in this era, new challenges arise for organizations, necessitating the designing and implementation of new systems that will scale easily to adapt to data needs and workloads. There are two approaches to scalability: scale out or scale up.

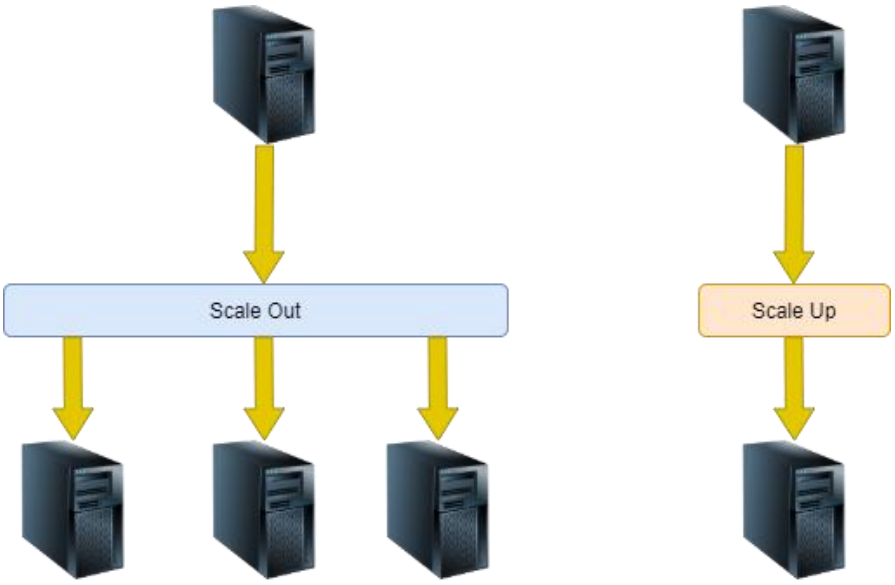

Figure 1. Scale Out and scale Up

Scale-Up

It is also known as vertical scaling. This scaling is all about the upgrading of hardware. Organizations scale up to increase their computing capacity by installing additional resources such as hardware or central processing unit (CPU). For example, an organization can buy a server with better processing capabilities and RAM when the hardware supporting its applications cannot sustain the growing workloads. Scale-up is easier to control since you decide on all hardware upgrades but it is also expensive since you have to buy more powerful hardware (central processing unit, disk, etc.). Note that scale up in relation to fault

tolerance presents more risk because in case of hardware failure there is the danger of losing data completely or facing hardware outages.

Scale-Out

It is also known as horizontal scaling. With horizontal scaling, machines with lower performance are added to a system to ameliorate its storage and computing capacity. For organisations, implementing scale-out will be cheaper because they will not be buying expensive high-performance machines, but instead machines with lower performance. Note that scale-out is better in terms of fault tolerance because it has mechanisms that put standby nodes or servers to specific services and perform data replication across different servers to ensure that in case of outage of some servers, data will not be lost and the services will still be available. Even though scale-out offers better fault tolerance, when it comes to debugging and finding the node(s) that cause problems in the system in case of failure, it will take more time to study logs and find the node(s) causing the problem due to the vastness of system. Apache Hadoop is a very successful scale-out open source project. Within Hadoop, the storage and computing capacity can be increased by simply adding new nodes/servers to the system in place.

Towards the end of the '90s, the constant adoption of mobile devices and new web technologies introduced novel challenges for both storage and processing of the large data quantities generated. This eventually leads to the rise of the concept called "Big data" [16]. Traditional data warehouses could not handle analytics on this huge data and more scaling-up computer hardware turned out to be very expensive. The solution comes in the form of a new distributed system paradigm [17].

Distributed systems are simply networks of a large number of attached nodes or entities connected through a fast local network [18]. This new paradigm required the development of adequate software and techniques to process data. thus, giving rise to the concept of big data analytics [16]. Big data analytics can be viewed as a sub-process in the overall process of insight extraction from big data [19].

It is common to describe the challenges introduced by big data within the so-called "V's of big data": volume; velocity; variety; and veracity. Volume has to do with the ever-growing quantities of data which can go beyond terabytes and petabytes. Variety refers to the fact that data comes from different sources such as computers, websites, and Internet of Things devices but also can come in multiple formats. Velocity has to do with the speed at which data is produced and needs to be processed. Veracity refers to the level of trust we have in the data or the accuracy of the data, which often requires to be analyzed in real-time.

The technology field of Big data is a fast-change ecosystem. New solutions are constantly in order to address and meet the, also, fast pacing evolving challenges associated with the V's of big data. Big data

up till now, does not have a unique universal definition, but different institutions and researchers provided different definitions to Big data. Some definitions of big data include:

- Gartner defines big data as a "high volume, velocity and/or variety information assets that demand cost-effective, innovative forms of information processing that enables enhanced insight, decision making, and process automation". [20]
- Schroeck [21] defines big data as a combination of Volume, Variety, Velocity and Veracity that creates an opportunity for organizations to gain a competitive advantage in today's digitized marketplace.
- Microsoft [22], defines big data as the process of applying serious computing power, the latest in machine learning and artificial intelligence, to seriously massive and often highly complex sets of information.
- Big data can also be considered as datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze. [23]
- Big can also be seen as the data sets and analytical techniques in applications that are so large and complex that they require advanced and unique data storage, management, analysis, and visualization technologies. [24]
- Big data is data that exceeds the processing capacity of conventional database systems. [25]
- Another definition by [26], indicates Big Data represents the Information assets characterised by such a high volume, velocity and variety to require specific technology and Analytical Methods for its transformation into value.

2.5 Distributed Computing Systems

A distributed system has diverse components located on different machines that communicate together and coordinate operations to appear as a single system in front of the user [27]. Within a distributed system, a machine can be a computer, virtual machine, container, physical server and any other node that can connect to the network, have local memory and communicate by passing messages [28]. Nodes refer to distinct entities in a distributed system.

For a system to qualify as a distributed system, it must have the following minimum characteristics: The components of distributed systems fail independently from each other; all components run concurrently and each component at its own clock, hence there is no global clock. With the evolution of mobile devices,

social media and sensors, the quantity of data has considerably increased. Since the 1980s, Relational databases(RDBMS) have been one of the most successful database
technologies. That notwithstanding, even with its solid technological growth, the relational database has failed to scale with the growth of data [16]. Even though there have been technological advancements over the years, relational databases still face challenges when it comes to scalability. With relational databases developed to support tabular data which is more structured, they face a challenge when it comes to dealing with semi-structured and unstructured data. The applications built using RDBMS technologies either have failed to perform better with increased data or the cost of running and maintaining the infrastructure to keep the application performing has grown exponentially [16]. Luckily, a new generation of storages was developed using distributed computing concepts and paradigms to help solve the limitations of relational databases. In the next paragraphs, different storage solutions following the distributed system paradigm will be further explained.

Distributed Storage

Even though No-SQL came to solve the needs for storing and managing large quantities of data and varying structure, not every application actually needs a database for storing and managing their data.
Previously, when data quantities of documents, images, videos were not large, they could be stored in the file system and processed with domain-specific tools such as text parser and image processing software. But as the size of data captured in these forms (i.e. documents, images, and videos) increased, it became difficult to store and manage these data in a single node computing system [16].
In the past, data stored within a local file system was processed using a computing system with a single node. But later on, better storage called RAID (redundant array of independent disks) storage, that was adapted for storage of large scale data in file systems. One advantage of RAID is that it had a failover mechanism, but only a single node could be used to process data. Even with this, data volumes still increased and processing data on a single node took longer times or was even impossible. This led to the introduction of distributed file systems, whereby data is distributed across multiple local hard disks with each being associated with a separate computing node [16]. With this approach, each node processes the data stored on its local disk which enables parallel processing in such a way that large quantities of data can be processed at the same time and shorten processing durations. Hence the more nodes are available within a system, parallel processing can be leveraged to accelerate data processing. This form of parallel storage and processing of data led to the development of different distributed file systems such as the

Google File System, Hadoop Distributed File System and other frameworks such as Map-Reduce and Apache Spark [16]. HDFS will be described better below.

Hadoop Distributed File System (HDFS)

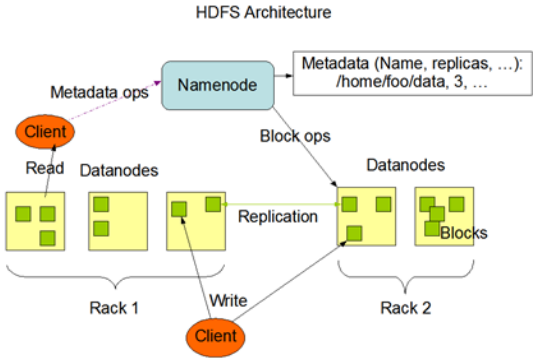

Figure 2. HDFS Architecture [47]

It is a type of distributed file system that spreads across multiple nodes, with each of them having their own local regular operating system for example Linux, on top of which HDFS is deployed. Within HDFS, the metadata and actual data files are stored separately, with actual/application data being stored on servers called DataNodes while metadata is stored on servers called NameNode [29]. The DataNode in HDFS does not have any individual failover mechanism such as RAID. Rather the file content is replicated on multiple DataNode for reliability. This has the advantage of data being local to the node, where the computation will be carried out, which in turn reduces the overhead associated with data transfers between the nodes for computational purposes [16]. Inodes are used to represent directories and files in the NameNode. Inodes record different things such as access and modification time, permissions, namespace and disks space quotas. The file content is split into blocks known as HDFS blocks. An HDFS block has a size of usually 128 MB, but maybe larger. Note that HDFS blocks are replicated on multiple DataNodes. The NameNode maintains the namespace tree and the mapping of the HDFS blocks to DataNodes (the physical location of the HDFS block) [29].

HDFS has an API that provides the locations of a file block, which enables distributed programming like Map-Reduce framework to process data in a node locally where the data is located [29].

Distributed Computation

Traditionally, distributed and parallel computing relied on synchronization and locking. Nevertheless, locking data and synchronization across multiple processes raises huge overhead. More so, traditionally, parallel and distributed computing handled computation independently from the data. It was assumed that data resided in a database or other storage system that is also accessible by many computing nodes.

The data is locked and processed in these nodes through parallel processing. In addition to the over head generated by locking, such an approach adds a lot of overhead in the transportation of data from the data node (where the data resides) to the processing node (the node processing the data). [16]. The Map-Reduce framework is a parallel programming framework that provides solutions to the issues associated with parallel and distributed computing.

Map-Reduce in Hadoop

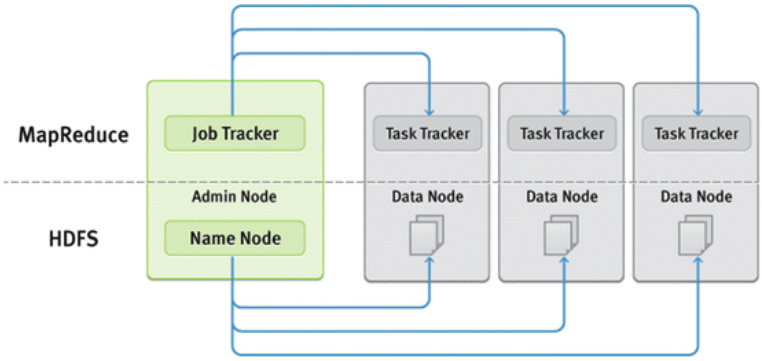

Figure 3. Map Reduce on HDFS [16]

The figure above showcases the architecture of Map-Reduce on HDFS. The Job Tracker breaks down the job into multiple tasks and assigns them to various nodes. The task trackers are responsible for task completion. Likewise, the Job tracker and the Name Node of HDFS can coexist in the same node. This enables the task tracker to process local data without transmitting the data from one node to another node. Additionally, the job tracker distributes the jobs in such a way that the task tracker processes only the local data. In Hadoop 2.0, the job tracker was replaced with YARN, a separate software component to manage tasks. The programming framework of Map-Reduce is based on considering data not as a single unit, but as a collection of multiple units [16]. With Map-Reduce, collections are considered as Maps. Map-Reduce performs data computation or processing in three steps including Map, followed by Shuffle & Sort and finally Reduce.

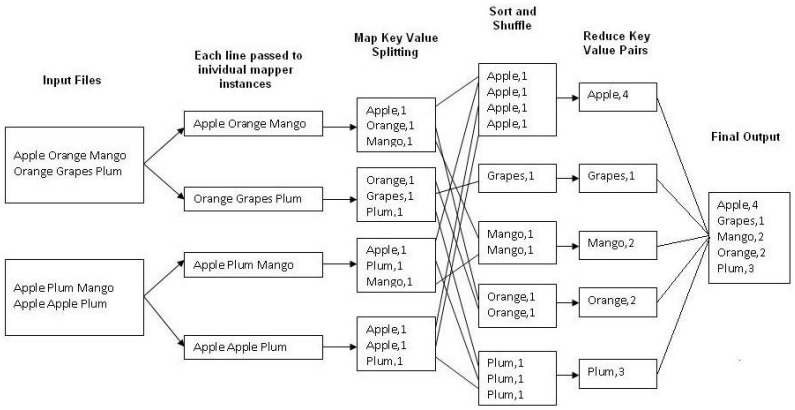

Figure 4. Map-Reduce word count example [30]

Above is an example of Map-Reduce computation where the input is a set of files. Each file is split as a collection of lines and the collection of all the lines is considered as a map, where the key is the location of the line and the value is the line. The map is the input of the Map step in the map-reduce program. map-reduce uses only the values in the input map.

The map step initiates different programs and each of these programs is called mappers. Each entry of the input collection is input to each mapper. The mapper program splits the line into words and creates a map of (Word, 1), where 1 is the count of the word in that line. The mapper program then sorts and shuffles this map. Likewise, with the help of sorts and shuffle, all the mapper programs send the entries (i.e. the count) associated with the same key (i.e. same word) to the same reducer.

When the reducer receives the values related to a key(word), it then sums up all the individual counts for each specific word and then writes the results to HDFS. Note that each reducer write is independent of other reducers and each write is kept as a separate file in HDFS. This results in multiple output files created by the map-reduce program, all running on top of HDFS. [16]

Apache Spark

Apache Spark was introduced shortly after the introduction of Hadoop. Apache Spark started as a research project by Matei Zaharia at the University of California, Berkeley AMPLab in 2009 and open-sourced in 2010. Spark's foundation came from Map-Reduce but got ameliorated, providing a robust, generalised framework for distributed computations on big data. Apache Spark as an open-source project was handed to the Apache Foundation in 2013 and since February 2014, Spark is a top-level Apache project. When it comes to big data, one map-reduce program is not enough to perform computation for analytic purposes. Most often a series of map-reduce programs will be needed to perform analytic computations, with the output of sequential map-reduce programs being the input to other map-reduce programs. Most often

the input to map-reduce programs is taken from distributed storages such as HDFS, No-SQL databases, etc. and the output from map-reduce programs are written to HDFS [16].

This constant read and write to HDFS over the long term becomes inefficient, generates computation overhead and its time consuming which over the long run can cause more problems like slowing analytics and delaying data processing and reporting. With the rise of machines with better memory and processing capabilities, Spark came in to make use of those to offer faster and more efficient in-memory processing capabilities. Unlike Hadoop, Spark makes use of memory to avoid constant read and write to the map-reduce workflow. Spark makes use of RDD to store collections of data.

RDD variables can be considered as collections of data that reside in memory and also span across different computers. This is advantageous because, the collection is distributed across multiple machines and processing of the collection can be done in parallel on all these machines, where each machine does the computation on its local memory. In addition, the distributed map-reduce processing in the case of the spark is done on RDD (Resilient Distributed Datasets) which makes it faster than map-reduce on HDFS. [16]

Apache Spark makes use of multithreading. Multithreading is the ability of a single core in a multi-core processor or central process unit (CPU) to execute multiple threads concurrently, appropriately supported by the operating system. [16]

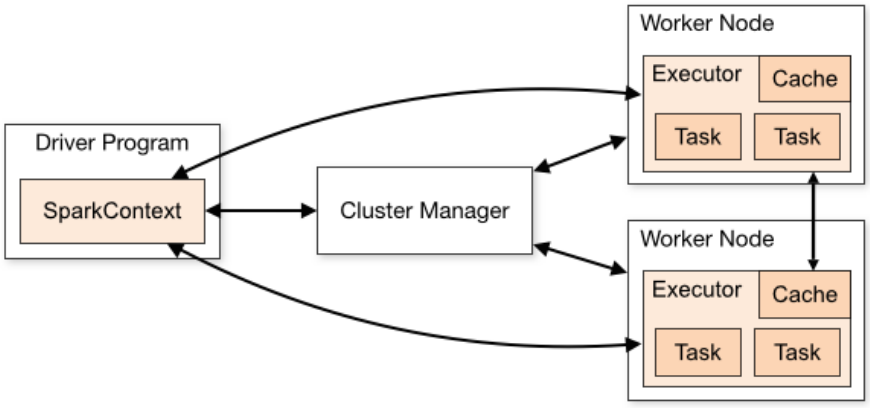

Figure 5. Spark multithreading model
Source: Retrieved from spark.apache.org

Spark applications run as an independent set of processes, coordinated by the SparkContext object in the Spark driver program within a cluster. In order for the Spark context to run on a cluster, it has to connect to different types of cluster managers. It can connect to the standalone cluster manager of Spark itself known as Mesos or another alternative known as YARN (Yet Another Resource Negotiator)) [31]. These cluster managers allocate resources across different application on the cluster. Once connected, Spark

acquires executors on nodes in the cluster, which are processes that run computations and store data for applications. Next, Spark sends the application code to the executors. Then the SparkContext sends the tasks to the executors to run. Each application gets its own executor processes, which stays up for the duration of operation of the entire application and run tasks in different threads. The advantage of this is that it isolates applications from each other, on both the scheduling side (that is, each driver schedules its own tasks) and executor side (that is, tasks from different applications run in different Java Virtual Machines). [31]

Apache spark rapidly became a go-to option for data enthusiasts and practitioners because it was easier to use, providing more functionalities that increased its utility and broadened its appeal, performed better on benchmark tests and supported streaming tasks. Spark's accessible interactive mode enabled data practitioners to perform exploratory data analysis(EDA) on large data sets, going beyond traditional Map-Reduce ETL jobs they had done before. Spark enabled training machine learning models at scale, querying large data sets using SQL, and real-time data processing much faster-using spark streaming. Additionally, since the early days of Spark, its popularity has grown, and it has become the go-to standard for big data processing, due to its sizable committed community members, dedicated and passionate open source contributors.

Nowadays, many data lake architectures use Spark as the processing framework that enables data teams to perform ETL, curate data and train machine learning models. In terms of speed, Spark runs an application in a Hadoop cluster up to 100x faster in memory and 10x more quickly when running on disk. Spark accomplishes this by reducing the number of reads and write operations to disk. Note that Spark stores the intermediate processed data in memory.

Spark has built-in APIs in R, SQL, Scala, Python and Scala, enabling data teams to write applications in different languages. Spark has up to 80 high-level operators for interactive queries. Besides, beyond supporting 'Map' and 'Reduce', Spark also supports SQL queries, streaming data, Machine learning and Graph algorithms.

The general execution engine for Apache Spark is called Spark Core. Spark core helps to reference data in external storage systems and provides in-memory computing. Additionally, Spark SQL introduces a data abstraction called Schema Resilient Distributed Datasets(SchemaRDD), that supports unstructured, semi-structured and structured data. Spark SQL also allows interaction with data on data frames which organises data in a tabular format.

Spark streaming leverages spark's core fast scheduling feature to perform streaming analytics. Spark ingests data in mini-batches and performs resilient distributed dataset transformations on the mini-

batches of data. Also, MLlib stands for Machine Learning Library. It is a distributed machine learning framework that seats on top of spark and includes some libraries and algorithms such as clustering, classification, etc. Additionally, GraphX is a distributed graph-processing framework on top of Spark.

Distributed Messaging Software

There is nothing like a single node system within a big data system, each component is considered as a cluster (one or more node(s) that take care of data distribution and computation). In such a scenario, creating and managing one to one communication becomes a challenge [16]. In order to avoid this challenge, big data messaging software or frameworks were introduced over the years such as Kafka, Flink and RabbitMQ. RabbitMQ is a message-passing software used to manage streaming data since pre-big-data days. From foundation, RabbitMQ had a single server system, but with the advent of big data, it has incorporated clustering in its architecture. The most popular message-passing system in RabbitMQ is the pub-sub(publication-subscription) system. Within the publication-subscription system, there are message publishers that form a group and publish messages with different subjects and on the other hand, there are message consumers that form a group and consume messages from the different subjects. RabbitMQ can process messages in a range of 20 to 30,000 per seconds and its strength is on routing [32].

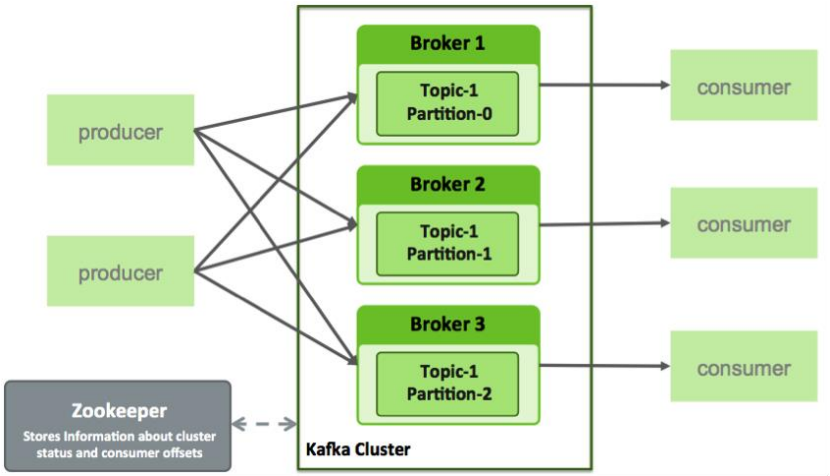

Figure 6. Kafka Cluster [33]

Apache Kafka is a clustered stream data processing software. Due to its inbuilt clustering technology, a Kafka cluster can process one hundred thousand to a few million messages per seconds. A Kafka cluster consists of multiple partitions and multiple servers. Each partition has one server which acts as the "leader" and zero or more servers which act as "followers." The leaders handle all read and write requests for that partition while the followers passively replicate the leader. Whenever the leader partition fails, one of its followers automatically replaces it by becoming the new leader. Each server acts as a leader for

some of its partitions and a follower for others, so the load is balanced across multiple servers. The strength of Kafka is that; it can consume massive volumes of stream data [34].

2.6 Data Integration

Regarding big data, as per Forrester Research's Q1 2016 report, data preparation and data integration technologies are within the scope of intense survival and growth phases, respectively. The ultimate goal is to make data flow faster and seamlessly across multiple sources in data storages and provide strong bases to enable analytics or business intelligence.

Besides, data integration requires evaluating data across many storages with different business needs to produce a centralised master data management system among other operational systems. Data warehouses or data lakes usually engage extraction, transformation and load processes to manipulate the transactional and functional structure of the data.

Moreover, a normal ETL process might be useful in managing structured, batch-oriented data that is up to date and within scope for organisational insights and decision making. On the other hand, dealing with stream data necessitates a different model and significant tweaking to the ETL process [35] where low latency, high availability, and horizontal scalability are vital features that need to be addressed in real-time or near real-time environments [36].

Extraction, Transformation and Loading (ETL)

The ETL process is a representation of data movement and transactional processes from the extraction of multiple data storages or sources, for example, transforming the data into a conformed format to send the data to target systems such as data warehouses or data lakes. The ETL process is usually applied in data integration, data migration, data staging and data management. The figure below illustrates the ETL process flow in a data lake environment. The flow involves accessing data from a source system, then ETL processing, that includes cleaning, integrating and staging the data before sending the transformed data to target systems.

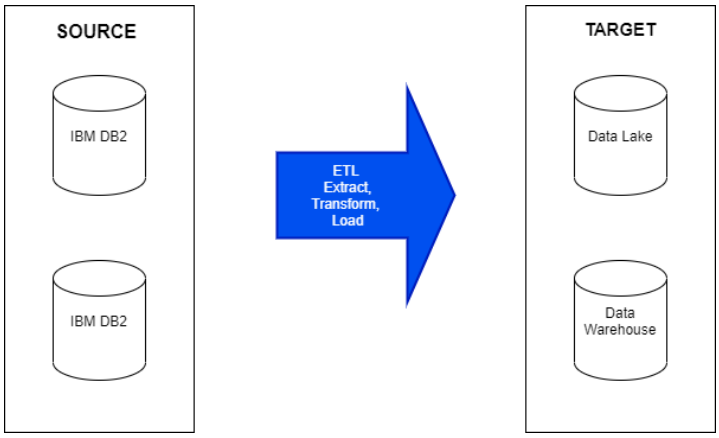

Figure 7. ETL system in a data lake or data warehouse environment

ETL systems or processes continuously need to support and evolve with the increase in data volumes, endless streaming, an increasing variety of data types and sources, requests for analytics and real-time user requirements, availability of new technologies and powerful tools [16].

Features of near real-time environment

The main features of the near real-time environment include its high availability of streaming data, horizontal scalability for performance improvement and low latency of intervals between transactions. Every operational system needs to address these features as it can limit the functionalities of the system if not appropriately handled.

Concerning high availability, streaming data is always available in constant flow within seconds or milliseconds. Streaming data is sensitive by nature, and the slightest disruption will affect operations. Replication and distribution are vital considerations to guarantee fluidity of the data collection process, lost data can be recovered and always available when called, even when faced with outages, receiving overloads of throughput or data loss [36].

Furthermore, on low latency, the speed requirement in delivering the most recent data to meet business demands is called data latency. Compared to the periodical requirement in batch data, the time between the events when data arrives and when data is made available to the user for near or real-time data is almost instant and has low latency [36]. The traditional ETL process was designed to satisfy batch processing where data refreshment occurs during non-peak hours, of which operational activities can halt temporarily. Hence could not produce accurate results for near or real-time analytics where stream data flows continuously.

Moreover, horizontal scaling is the viable approach for improving performance and alleviating the risk of interruption of seamless data flow [36]. Horizontal scalability adds separate independent servers to a cluster of servers handing different tasks and needs only a little coordination between systems. Horizontal

scaling offers more potential for expansion and less risky measures compared to vertical scaling, in which boosting its performance is limited within the capacity of a single machine. In vertical scaling, if the machine has outages, the data flow is disrupted, and the system can potentially lose pertinent data.

Challenges within the ETL stages for near real-time environments

With the constant flow of most recent data in large volumes and real-time reporting with near or real-time environments, traditional ETL systems need major rework and remodelling to support the requirements [38]. Highlighted below are challenges of each ETL process step and possible solution approaches.

Extraction

This process involves identifying dimensional data attributes from different databases, capturing change data and getting data from the source location [38]. Streaming data like click streams, event logs, and sensors are regularly and continuously changing. Stream data challenges the way change data is captured for constant update and loaded without disrupting normal operational activities [39] [40]. The figure below identifies challenges associated with the extraction step in the ETL process within a near real-time environment and solutions approaches.

Feature	Extraction Stage	
	Challenge	**Solution Approach**
High Availability	a) Heterogeneous Data Source. b) Backup Data. c) OLAP Internal Inconsistency.	a) Stream Processor, Semantic web technologies toolkits. b) Server for replication; Log-based Change Data Capture. c) Snapshot; RTDC (Real-Time Data Cache); Layer-based View; RODB(Real-Time Operational Database).
Low Latency	a) Multiple Data Source Integration. b) Data Source Overload.	a) Combine change data capture, stream processor and data integration tools. b) Update significance and record changed method; Special format for Change Data Capture log.
Horizontal Scalability	-	-

Table 1. Challenges and solutions in the extraction stage for near real-time environment [38]

Transformation

Transformation is the process whereby data is cleaned and conformed into a predetermined format, shared across different organisational platforms and needs [38]. Constant refreshment of transactional data occurs in near-real-time data, hence the frequency at which the master data updates is higher than that of batch data. Also, the quantity of data carried into the target system after transformation is smaller and constant, that makes the transformation process efficient by processing smaller amounts at a more frequent rate [41].

Feature	Transformation Stage	
	Challenge	**Solution Approach**
High Availability	-	-
Low Latency	a) Master Data Overhead. b) Intermediate Server For Aggregation.	a) Master data cache and database queue. b) ELT (Extract Load Transform).
Horizontal Scalability	Separate Server For Aggregation	ELT (Extract Load Transform).

Table 2. Challenges and solutions in the transformation stage for near real-time environment [38]

Loading

In this step, transformed data or metadata is sent to the target storage system, for example, a data warehouse or data lake. The biggest challenges are to maintain maximum performance during Online Analytical Processing(OLAP) or analytical process to avoid overlap while loading data [41] [40] and also, due to OLAP or analytics internal inconsistency.

Feature	Loading Stage	
	Challenge	**Solution Approach**
High Availability	OLAP Internal Inconsistency	Snapshot; RTDC (Real-Time Data Cache); Layer-based View; RODB(Real-Time Operational Database); Dynamic mirror
Low Latency	Performance Degradation	Staging Table; Multi-Stage Trickle & Flip

Horizontal Scalability	OLAP Internal Inconsistency	Staging OLAP outside data warehouse update period; CR-OLAP(Cloud-based Real-time OLAP system)

Table 3. Challenges and solutions in the loading stage for near real-time environment [38]

2.7 Delta Lake

According to delta.io, Delta Lake is an open-source storage layer that brings atomicity, consistency, isolation and durability transactions to Apache Spark and big data workloads [48]. Databricks released delta lake in 2019 and it is presently open-sourced with the Linux Foundation project. Delta lake came in to solve shortcomings of data lakes. There are two main challenges associated with data lakes, including data reliability and Query performance which delta lakes come in to solve.

Data Lake data reliability issues and Delta Lake solutions

In the absence of the right tools, data lakes can rapidly face data reliability problems. In the next paragraphs, the instigators of data reliability issues on a data lake and how delta lake comes in to solve the problem are better explained.

Reprocessing data continuously due to broken pipelines or corrupted data in a traditional data lake are significant issues. Pipeline breaks mostly occur due to hardware or software failures during the data writing phase in a data lake when the job does not complete. When this happens, data teams spend lots of time and energy deleting corrupted data, verifying the correctness of the remaining data and setting a new write job to fill in any loopholes in the data. On the other hand, delta lake solves the reprocessing issue by making the data lake transactional, hence ensuring that every operation performed on the data lake is atomic. Atomicity will ensure that either the operation succeeds entirely or fails. Data teams, in turn, will not spend lots of time reprocessing data due to broken pipelines or failed writes.

Quality enforcement and data validation are also issues in data lakes. Data validation is vital in data applications because, in its absence, there is no way to verify whether something is inaccurate or broken in the data, as opposed to traditional software applications where something wrong is easier to find, for example, a broken website which will display an error message. Data quality problems go unnoticed in data applications. Sometimes these corrupted data can only be identified after pipeline breaks, which becomes painful to solve depending on the critical level. On the other hand, delta lake's features of schema enforcement and schema evolution help to manage data quality. Schema enforcement and schema evolution, allows data teams to specify a schema and enforce it, also change a table's current schema to cope with changing data over time, respectively.

Bulk updates, merge, and deletes are also data reliability challenges in data lakes. Since data lakes can contain large amounts of data, organisations need to reliably perform an update, merge and delete operations on data stored in data lakes, make sure the data is up to date at every point in time. With traditional data lakes, it can be extremely challenging to perform delete, merge and updates operations and confirm everything of each was successful since there is no mechanism to ensure data consistency.

In recent years, not being able to perform delete, merge and update effectively has become a considerable burden for organisations since current regulations such as CCPA and GDPR require organisations to delete all of a customer's information upon request. Traditional data lakes face two challenges, making this delete request. Organisations need to be able to query all the data in their data lake using SQL, also delete any data related to that customer on a row-by-row basis, that is something traditional analytics engines are not in-built to do. On the other hand, the delta lake solves this by enabling data teams to query the data in data lakes using SQL easily. Then perform updates, merge and delete on their data with a single command, thanks to delta lake's ACID (atomicity, consistency, isolation, durability) transactions. Additionally, Delta lakes combine streaming and batch sources and sinks, that enables the creation of a single flow of data that allows users to focus on data quality [42] when implementing data movement processes, hence ensuring top data quality.

Data Lake query performance issues and Delta Lake solutions

The second major challenge associated with data lakes is query performance. Query performance is a significant driver of user satisfaction for data lake analytics tools. For data teams performing exploratory data analysis using SQL, rapid response to common queries is essential. Since data lakes hold a large number of files and tables, the data lake query engine should be optimised for performance at scale.

Small files in large numbers within a data lake instead of larger files can slow down performance considerably due to limitations with Input/Output throughput. Delta lakes solve this by using small file compaction to group small files into larger files, optimised for reads.

Furthermore, a slowdown in query performance can happen due to repeated access of data from storage. Delta lake solves this by using its caching feature, to selectively hold important tables in memory, to ensure the quicker recall of the tables. Delta lake also uses data skipping to boost read throughput by up to 15x, to avoid processing data that is not relevant in a given query.

Moreover, on modern data lakes using cloud storage, files deleted from the data lakes can remain for up to 30 days, which creates overhead that slows query performance. Delta lake has a vacuum command to delete files that are no longer needed permanently.

Likewise, to facilitate query performance, the data lake has to be appropriately indexed and partitioned along the dimensions by which it is most likely to be grouped. Delta lake can create and maintain indexes and partitions, better optimised for analytics.

Also, as the data lake size grows to petabytes or more, a bottleneck might not be the data itself, but the metadata that accompanies it. Delta uses Spark to offer scalable metadata management that distributes its processing, just like the data itself.

The delta lake's architecture is called the delta architecture, and it has three different stages of data enrichment including Bronze (raw zone), Silver (prepared zone), and Gold (curated zone).

The bronze zone is where all raw data is dumped and stored there for the long term. The silver zone is where basic data cleaning and aggregation happen, and the gold zone contains curated data that is ready for consumption by stakeholders, applications, and machine learning models. Also, this setup enables faster reprocessing of data because since data is available on all zones, in case data has to be reprocessed within a zone, tables are deleted on that zone and processing is run on the previous zone. Hence avoiding going through the entire data loading part again.

2.8 Modern Data Storage Solution

With the amount and variety of information growing exponentially due to digitisation, social media, and internet of things (IoT), organisations are in high need to develop a data-driven culture. Traditional data storage solutions are getting quickly outdated due to their inflexible nature and inability to combine huge volumes of different data types.

Nowadays organisations need to have a more proactive approach when it comes to decision support and decision making but face a significant challenge due to the lack of access to accurate, relevant, and reliable information on a timely basis [13]. Traditional organisations rely on data warehouses. However, data warehouses become a burden as they cannot cope with the market pressures suffering from deep limitations in adapting to new data formats requirements and technological developments. The above brings up a bigger question as to whether 'Old data warehouse' still meets the requirements of their users' needs.

A modern data platform is a future proof architecture for analytics with components supporting a modern data warehouse, machine learning and artificial intelligence development, real-time data ingesting and processing [43]. A modern data platform is vital to help organisations overcome the challenges linked to traditional data storage solutions. In this new decade, organisations require a data platform that will enable them to adapt to continually evolving business needs and manage data growth. Organisations must be agile and proactive, with standardised processes and a single source of truth to better support

decision making. Additionally, the MDP must run on state-of-the-art software supported by high-performance hardware.

Databricks as a Modern Data Processing and Storage solution

Databricks enables organisations to bring their data warehouses or data platforms into a modern era. Note that, there are other data platforms available today such as Cloudera, but the focus in this work will be on Azure databricks. Azure Databricks is an analytics data platform enhanced for the Microsoft Azure cloud [44].

Azure Databricks offers collaboration across the full data and machine learning lifecycle with the Azure Databricks workspace. It has collaborative notebooks which enable data teams to access and explore data quickly. Additionally, it offers preconfigured machine learning environments with machine learning frameworks such as PyTorch, Tensorflow, Sci-kit Learn, Keras and XgBoost. Machine learning experiments can be tracked and shared, runs reproduced, and models can be managed collaboratively from a central repository, from experimentation to production. Azure Databricks also supports different business intelligence tools such as Qlik, Microsoft powerBI, Looker and Tableau.

That notwithstanding, Azure Databricks includes delta lake, that brings data reliability and scalability to an existing data lake on the Azure cloud. Databricks offers simple batch and stream data processing on auto-scaling infrastructure powered by highly optimised apache spark. Also, databricks offers enterprise-level security, which offers native protection to safeguard data where it lives and creates compliant, private and isolated analytics workspaces across thousands of users and datasets.

Organisations can audit and analyse all the activity in their account and set policies to administer users, control their budget on Azure cloud. Besides, organisations can run and scale their most mission-critical data workloads on Azure cloud, with the ecosystem integrations for continuous integration and continuous development and monitoring. The modern data platform designed with Azure Databricks provides answers to the five challenges associated with traditional data warehouses/platforms including rigid structure, complex architecture, slow performance, outdated technology and lack of data governance, covered previously in this section.

Azure Databricks, through delta lake, stores, and analyses data no matter its source, form and structure, enabling data teams to perform ETL or ELT process the data across their data platform. With ETL, data can be processed before loading on the delta lake area, while with ELT, the data is loaded directly in the delta lake area before it undergoes a transformation.

Besides, Azure Databricks offers an open and simplified architecture on the cloud to tackle data governance issues. Databricks combines data within a single system, enabling situational data analysis that reduces data movement and computation resources wastage.

Furthermore, with the rapid rise in demand for speed in data preparation and consumption processes, Databricks offers advanced analytics and speedy transactions throughout the data processing life cyde. Spark's in-memory architecture lowers disk bottlenecks and accelerates performance to provide a rapid, and accurate response. Azure Databricks also helps a lot to accelerate data integration from different sources and to load them on the target systems on and out of cloud environments.

Moreover, traditional data platforms/warehouses have little or no ability to support big data and IoT workloads. But, Azure Databricks comes in to solve those challenges and boost performance. Azure Databricks is hosted on Microsoft Azure cloud, enabling a 'pay as you go' system for organisations. This helps to reduce capital expenditure with organisations not buying physical assets and reduces financial risk. Azure Databricks through delta lake brings ACID transactions to data lakes on Azure and enables organisations to better curate data available on their data lakes.

Additionally, Azure Databricks provides comprehensive security and auditing functionalities. Also, solutions that provide access control to data within teams and organisations. Databricks provides data lineage and time travel that ameliorates traceability of actions performed by users.

All the above put Databricks on Microsoft Azure cloud, a preferred solution to build a modern data platform on the cloud.

3. Tools and Technology

The MDP project had different technology components. The tools used during the internship to build the delta lake and ETL pipelines will be explored in this chapter, to give a more detailed overview as to why the financial institution chose these specific technologies.

Azure Cloud

Microsoft Azure cloud is a continually evolving set of cloud services to help organisations solve their business problems. The Azure cloud offers the freedom to build, manage, and deploy applications on a massive, global network using different tools and frameworks. The Azure cloud technologies used in MDP were Azure Data Lake Storage Gen2, Azure Cosmos DB, Azure Databricks with delta lakes.

3.1 Azure Data Lake Storage Gen2

Azure Data Lake Storage Gen2 is a group of capabilities focused on big data analytics, built on Microsoft Azure cloud blob storage. It is the result of combining the capabilities of Azure Data Lake Storage Gen1 and Azure Blob storage. Its components include features from Azure Data Lake Storage Gen1, such as directories, file system semantics, file-level security and scale, combined with low-cost, tiered storage, high availability and disaster recovery capabilities from Azure Blob storage.

Data Lake Storage Gen2 is built on top of blob storage and enhances performance, management, and security in the following ways:

- It is optimised because data does not have to be copied or transformed as a requirement for analysis. The hierarchical namespace in Azure data lake gen 2 enhances directory management activities compared to the flat namespace present in blob storage.
- It is easy to manage because files can be organised and manipulated through directories and subdirectories.
- It is highly secured because POSIX permissions can be defined on directories or individual files. Security is paramount because the client is a financial institution, it entails stricter data protection rules.
- Azure Storage is scalable by design, whether it is accessed via Data Lake Storage Gen2 or Blob storage interfaces. It can store and serve many Exabyte of data. This quantity of storage is available with its throughput measured in Gbps (gigabits per second) at high levels of IOPS (input/output operations per second). This scalability component allows the storages of the financial institution to scale with increasing data quantities.

- Another benefit of building Data Lake Storage Gen2 on top of Azure Blob storage is the relatively low cost of storage capacity and transactions. Unlike other cloud storage services, data stored in Data Lake Storage Gen2 is not necessary to be moved or transformed before performing analysis.

Other alternatives for data lake services provided by different cloud companies include:

i. **Amazon simple storage service (Amazon S3)**: It is a data lake solution developed by Amazon web services, a subsidiary of Amazon. Due to its virtually unlimited scalability, Amazon S3 can serve as a data lake. The storage can be increased seamlessly and non-disruptively from gigabytes to petabytes, and paying only for the storage used. Amazon S3 provides 99.999999999% durability.

ii. **Google cloud storage**: Another alternative to azure data lake gen2 is google cloud storage. It is a RESTful online file storage web service for storing and accessing data on Google Cloud Platform. Google cloud storage combines the scalability and performance of Google's cloud with advanced sharing and security abilities.

3.2 Azure Databricks

Another Azure cloud technology used in the MDP project was Azure Databricks. Databricks is an Apache Spark-based analytics platform optimised for Azure cloud. Developed with the creators of Apache Spark, Databricks is unified with Azure cloud to provide streamlined workflows, one-click setup and an interactive workspace that enables collaboration between data scientists, data engineers, and business analysts.

Azure Databricks is composed of the complete open-source Apache Spark cluster technologies and capabilities. Other alternatives to the azure data bricks services provided by different cloud companies include:

i. **Amazon EMR (Elastic Map Reduce)**: EMR is a managed cluster platform provided by AWS to help organizations run big data frameworks such as Hbase, Hive Hadoop, Spark, Hudi and Presto on the cloud to extract, transform and analyze large quantities of data. By using this framework, organisations can process data for analytics purposes and business intelligence workloads. Amazon EMR is used to extract, transform and load data into and out of AWS storages services, for example, Amazon Dynamo DB and Amazon S3 (Simple Storage Service).

ii. **AWS Databricks**: It is a unified data analytics platform for accelerating innovation across data science, data engineering, and business analytics, integrated with the AWS infrastructure.

iii. **Cloud dataproc**: Dataproc is a managed Hadoop and Spark service provided by Google cloud platform to enable data teams to process data both in batch and stream, query and run machine learning algorithms.

Dataproc automation helps to create clusters quickly, efficiently manage them and save money by turning off clusters when they are not in use. With less money and time spent on administration, data teams can focus on their jobs and data.

3.3 Delta Lake

Delta lake is a tool proposed by Databricks used in the MDP project. The promise by the creators of delta lake which is Databricks is that with delta lakes there is no more malformed data ingestion, difficulty deleting data for compliance, or issues in modifying data for change data capture. It accelerates the velocity that high-quality data can get into a data lake and the rate that teams can leverage that data, with a secure and scalable cloud service.

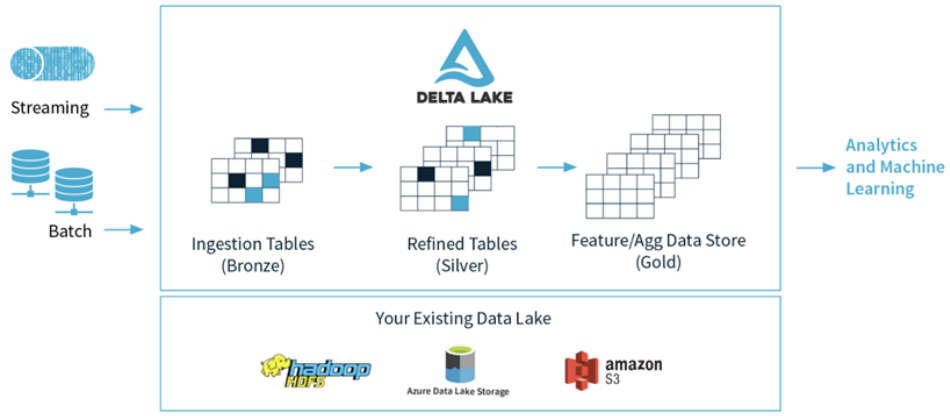

Figure 8. Delta Lake Layout

Source: Retrieved from delta.io

Below are the features of the delta lake that made it the go-to solution to resolve issues associated with typical data lakes for the financial institution:

- Numerous data pipelines can read and write data simultaneously to a delta lake. A data lake's ACID Transactions ensure data integrity with serializability, the most substantial level of isolation.
- Delta Lake provides data manipulation APIs to merge, update, and delete datasets. It allows the organisation to easily comply with the General Data Protection Regulation(GDPR)/California Consumer Privacy Act(CCPA) and simplify change data capture.
- It enables data lake schema specification and enforcement, ensuring that the data types are correct and required columns are present, and preventing corrupted data from causing data corruption.

- Delta lake enables time travel (Data Versioning). Data snapshots allow the organisation's data teams to access and revert to earlier versions of data to audit data changes, reproduce experiments or rollback bad updates.
- All the data in the Delta Lake is stored in Parquet format. The parquet format enables Delta Lake to leverage the efficient compression and encoding schemes that are native to Parquet.
- A table in a Delta Lake is a streaming source and sink, but also a batch table. Delta Lake enables the organisation to make changes to a table schema that can be applied automatically, without the need for cumbersome Data Definition Language.
- The delta lake transaction log records every detail about any change made to the data, providing a complete history of changes, for audit, compliance, and reproduction purposes.

So far, the delta architecture or delta lake has no direct competitor providing similar or more competitive features.

3.4 Azure Cosmos DB

Another Azure cloud tool used in the MDP project was Azure Cosmos DB. Cosmos DB is a globally distributed and multi-model database service created by Microsoft. Cosmos DB enables developers with a click of a button, to elastically and independently scale storage and throughput across any number of Azure regions worldwide. Developers can elastically scale storage and throughput, and take advantage of fast, single-digit-millisecond data access using their favorite API, including MongoDB, SQL, Gremlin and Cassandra.

Cosmos DB guarantees single-digit millisecond response times and 99.999% availability. The most significant advantage of Cosmos DB over all its competitors is that it provides four storage API's all at once within one database.

Other alternatives azure cosmos DB services provided by different cloud companies include:

i. **Amazon DynamoDB**: It is a document and key-value database created by Amazon web services that deliver single-digit millisecond performance at any scale. DynamoDB has the capacity to handle more than 10 trillion requests per day and can support peak request which reaches 20 million per second.

   DynamoDB supports ACID transactions to enable data teams to build business-critical applications that can quickly scale. Additionally, DynamoDB encrypts every data by default and provides fine-grained access and identity control on all the tables within the database. Developers can easily create full backups of terabytes of data instantly without any performance impact to tables on the database and recover at any point in time in the preceding 35 days with no downtime.

ii. **MongoDB Atlas**: It is another alternative to Azure cosmos DB. The centre of MongoDB Cloud is called MongoDB Atlas. It is a fully managed cloud database for modern applications. MongoDB's document model brings flexibility and ease of use to databases. Atlas is available in more than 70 regions across Amazon Web Services, Azure Cloud, and Google Cloud Platform.

3.5 Confluent - Schema registry

This tool enforces schemas and handles schema evolution. With the potential change of schema happening in the future and for the data pipelines not to break, the schema registry will intervene to ensure that does not occur.

Furthermore, Schema Registry provides a serving layer for metadata. It provides a RESTful interface for keeping and retrieving Avro schemas. It keeps a versioned history of all schemas, provides multiple compatibility settings and enables the schema evolution according to pre-configured compatibility settings. Confluent Schema registry has serializers that connect to Kafka clients to handle schema storage and retrieval for Kafka messages that have an Avro format [45].

An alternative to the confluent schema registry service is the Red Hat Integration service registry. It is a datastore for standard event schemas and API designs. It enables developers to decouple the structure of their data from their applications and to share and manage their data structure using a REST interface. The Apicurio Registry open source community project supports the development of a red Hat service registry.

Also, the service registry handles the Apache Avro JSON Schema, Protobuf (protocol buffers), OpenAPI and AsyncAPI data formats. Developers can configure rules for each artefact added to the registry to govern content evolution. Before sending a new version to the registry, all commands configured for an artefact must pass. The objective of these rules is to prevent invalid content from being added to the registry. An alternative to Confluent schema registry is Red Hat's Integration service registry. This registry from Red Hat provides a Kafka schema registry that helps to store Avro schemas.

4. Projects

At the beginning of the MDP project, every team member signed an NDA document with Accenture and their client (banking financial institution) which limits the sharing of information about the project, code or details of technology configurations done within the project. Examples of diagrams, code and configurations throughout this work, will be similar to those done during the MDP project.

The MDP project consisted of building a modern data platform for the banking financial institution on the Microsoft Azure cloud. The new data platform was going to be a set of integrated storages that will consume data from the company's legacy system, aggregate the data, structure it, and store it on azure cloud storage resources.

Within the MDP project, the Accenture team focused on the project section to migrate and transform data from the mainframe storage to azure data lake gen 2, which will serve as the data lake for the organisation. In addition to this, extract data from the data lake to load it on azure cosmos DB, which was going to be used to serve mobile applications developed by the banking financial institution.

Besides, we also performed code refactoring, unit tests, and pyTest on the MDP code within Databricks and locally on computers.

Timeline

Before starting the MDP project at the client site, between November and December 2019, the project team worked at the Accenture office where we received hands-on practical and technical knowledge on big data technologies related to the project, leadership, and communication skills to better prepare for the client work. This training had workshops, group projects, and presentations.

From January to June 2020, we worked directly with the client on their MDP project. From January to March, the team worked at the client site, but due to COVID-19, we had to work remotely from home after March.

The project structure followed the agile methodology, and Scrum framework was adopted. The Scrum framework helps teams work better together. The project had a product owner who served as the face of the client within the project and a scrum master who helped the team adopt the scrum methodology's best practices.

Furthermore, a product backlog helped to prioritise the project work. A product backlog is a list of work, prioritised for the development team, derived from the project roadmap and its requirements. The product backlog's priorities were defined by the team's product owner. Any work done by the team, was conducted within a sprint. Sprints are short, time-boxed periods when a scrum team works to complete a predefined amount of work. Sprints are at the centre of the scrum and agile methodologies, and getting

sprints right, helped the team deliver software with fewer headaches. Each sprint had a duration of two weeks.

In line with the above, to plan for each upcoming sprint, the team had sprint plannings. During the planning meeting, the team decided what work to focus on and how to accomplish it. The product owner and the team defined work based on the backlog items still left to do. Note that the backlog serves as the connection between the product owner and the development team. It is during this planning that each team member's work was defined based on his or her accomplishments of the previous sprint in completing tasks and future tasks that were left to do.

Once the sprint had started, the team had daily stand-ups, during which we all reported work progress. During this meeting, the team members highlighted the current challenges that were going to impact the team's ability to deliver the sprint's goal. Once a sprint was over, there was a sprint review. During the sprint review, each team member had the opportunity to present their work to stakeholders and teammates before sending to production.

Also, the team had sprint retrospectives that were done at the end of sprint cycles to spot areas of improvement for the next sprint. Note that all highly concurrent spark clusters on Databricks for this project were configured by the client's technology team, to ensure monitoring of resources and to enable team members to share computing resources of the same cluster. All team members wrote and organised their code in interactive Databricks notebooks.

4.1 Project 1 - Building Data pipelines to move data from mainframe storages, transform and load on the delta/data lake and cosmos DB.

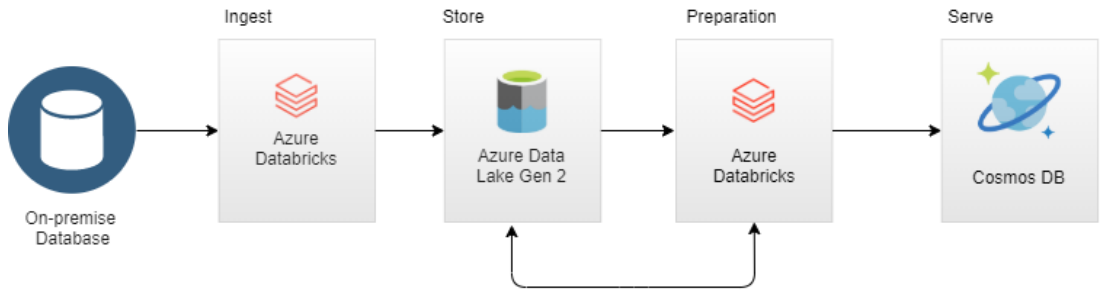

Figure 9. Modern data platform of the financial institution

The MDP of the banking financial institution had different areas including ingestion, store, preparation and serving layer.

The ingestion section focuses on the extraction of data from on-premise databases of the financial institution in streams with the help of Azure Databricks, then sending the data to Azure data lake Gen 2. Azure Databricks was chosen as an ingestion tool because it is an Apache Spark-based analytics platform

optimised for the Microsoft Azure cloud services platform. It supports both stream and batch data ingestions, hence can easily consume stream data from Apache Kafka.

Furthermore, the store layer helps with the storage of data on the data lake. The data lake used by the financial institution is Azure data lake gen 2 due to its capabilities dedicated to big data analytics such as file system semantics, directory, file-level security, also, due to its scalability combined with low-cost, tiered storage and high availability/disaster recovery capabilities.

Data preparation was done using Azure Databricks because it will help the financial institution to unlock insights from all their data and build artificial intelligence (AI) solutions. Azure Databricks provides interactive workspace and on-demand scalable clusters that enable collaboration between all members of a data team. Besides, Databricks brings in delta lake, to allow ACID transactions on top of the data lake. Additionally, Databricks has fully scalable, secured and managed clusters which teams can use flexibly to process massive amounts of data.

The serving layer used ComosDB because it is a globally distributed, multi-model database service for operational and analytics workloads. Additionally, it provides 4 APIs, namely, SQL, MongoDB, Cassandra, and Gremlin.

Project phase 1 - Data Migration from On-premise database to Delta lake raw(bronze) zone

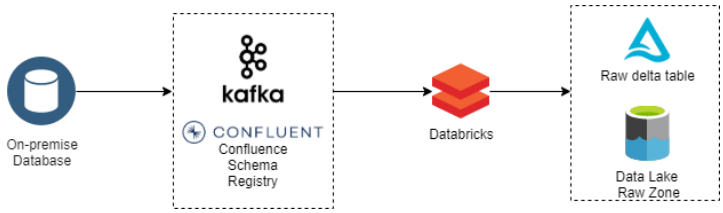

Figure 10. Data migration resources lay out

The objective of this phase was to ingest the data from the company's databases and send it to the delta lake raw zone.

The data extraction from the clients' storages happened in streaming mode. The data centre team of the client had to capture data in real-time from the databases as events. Apache Kafka was used as an event streaming platform by the financial institution and all the data from the databases were converted to Avro format and published to Kafka topics progressively during the project implementation.

Apache Avro is an Apache Foundation open-source project that offers data serialisation and exchange services for Apache Hadoop. Apache Avro eases the exchange of big data between programs written in any language. With the serialisation service, programs can efficiently serialise data into files or messages. The data storage is compact and efficient. Apache Avro stores the data definition and the data together in one message or file.

Furthermore, Apache Avro stores data definitions in JSON format and data is kept in binary form. An important point to note about Avro is that it provides strong support for data schemas changing over time — often called schema evolution. Avro can handle schema changes such as missing, added and changed fields; this enables old programs to read new data and new programs to read old data. Avro includes APIs for Java, Python, Ruby, C, C++, and more.

Additionally, data stored in Avro format can be exchanged between programs written in different programming languages such as C, Java, etc. The extraction of data from the databases occurred at different intervals during the project, and new fields were added to the Avro files with their schemas continually changing. The team had to ensure that the Avro files schema could change over time with the addition of new fields, without rebuilding the entire Kafka platform and avoiding breaking data pipelines. Additionally, Confluent's confluence schema registry came in as the perfect tool to help the data centre team implement schema evolution in the Kafka platform. To ensure that as new fields were added to the Avro files extracted at different intervals and sent to Kafka, the Avro schema registered in Kafka could evolve seamlessly without breaking the Kafka platform, taking into account new fields added.

Moreover, once the data was available in Kafka, we then used Azure Databricks to read Kafka streams, using built-in PySpark Kafka consumer configurations, that were re-adapted for the project use case. The subscription is on only one Kafka topic at a time.

Below is an example of PySpark code to read Kafka streams;

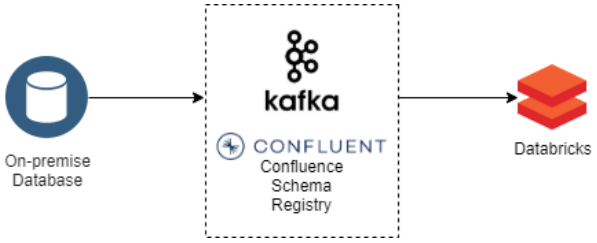

Figure 11. Kafka streams consumption layout

```
kafka_stream_df = spark.readStream.format("kafka")
                                  .option("kafka.bootstrap.servers", brokers_list)
                                  .option("subscribe", topic).option("startingOffsets", "earliest")
                                  .load().select(
                                  A.from_avro("key", key_avro, SCHEMA_REGISTRY_URL).alias("key"),
A.from_avro("value", value_avro, SCHEMA_REGISTRY_URL).alias("value"), "topic", "partition", "offset", "timestamp")


dataframe = kafka_stream_df.select(F.col("key").alias("key"), F.col("value").alias("value"),
                                  F.col("topic").cast("string"), F.col("partition").cast("int"),
                                  F.col("offset").cast("long"),F.col("timestamp"))
```

Figure 12. PySpark Kafka read stream

From the above code, the variable kafka_stream_df contains data from the Kafka stream. Its variables include:

- **Format("Kafka"):** indicates the file format. It is Kafka because we are reading data from the Kafka stream.
- **kafka.boostrap.servers**: The Kafka "bootstrap.servers" configuration is a comma-separated list of host and port pairs that are the addresses of the Kafka brokers in a Kafka cluster that a Kafka client connects to initially bootstrap itself. The brokers_list is a predefined variable that contains the name(s) of a broker(s).
- **Subscribe**: The topic list to subscribe. Only one of "assign", "subscribe" or "subscribe pattern" options is the Kafka source. The topic part is a predefined variable containing a topic name.
- **startingOffsets**: The start point of a query. It is either "earliest" which indicates to start from the earliest offsets or "latest" which indicates starting from the latest offsets, or a JSON string specifying a starting offset for each topic partition.
- **from_avro**: This helps to transform Avro data into a column. Apache Avro is a commonly used data serialisation system in the streaming data pipelines.
- **Key_avro**: Key of the Kafka message.
- **value_avro**: Value of the Kafka message.
- **SCHEMA_REGISTRY_URL**: variable with URL of the Schema Registry to define schemas to the topics.

The next part of the code above selects some elements from the Kafka stream and turns them to columns, stored in a spark dataframe. Then the spark dataframe is written as a delta table to the raw zone of the data lake. Below is an example of code.

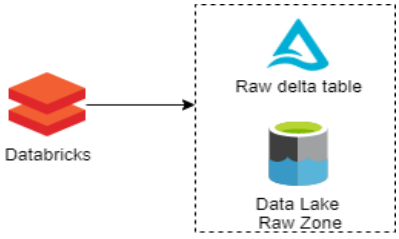

Figure 13. Writing data to the raw delta table on the data lake raw zone

```
dataframe.writeStream
         .format("delta")
         .outputMode("append")
         .option("checkpointLocation", checkpoint_location)
         .start(table_location)
```

Figure 14. PySpark write data stream to raw zone

The components include:

- **format("delta")**: The file format. The format is delta because it's a delta table.
- **outputMode**: mode in which data adds to the table. Structured streaming runs by default in append mode, but can also run in an update or complete mode.
- **checkpointLocation**: Output sink where the end-to-end fault-tolerance can be guaranteed, developers specify the location where the system will write all the checkpoint information. In the case of stream write failure, the ending location is stored so that once the stream starts, it will begin from the point it previously ended. The checkpoint location is a directory in an HDFS-compatible fault-tolerant file system. Checkpoint_location is a predefined variable which contains the checkpoint path on HDFS.
- **table_location**: the path to the location where the table will be stored. The path is the raw zone location on azure data lake Gen 2.

During this project phase, there were excessive waiting times for the client's team to write data to Kafka topics due to security scrutiny and the amount of data extracted that needed administrative approval. Additionally, for some of the data to be extracted from the client's legacy database, new API's had to be developed by the client's team. To ensure proper conversion of data to Avro format, that took some development time and increased waiting time for data to be available in Kafka. The delay sometimes affected the delivery of project work on time.

Project phase 2 - ETL processes to move data from delta lake to Cosmos DB

This phase of the project consisted of building the prepared and curated zone on the delta/data lake and sending data from the curated area to Cosmos DB.

Building the prepared(silver) zone

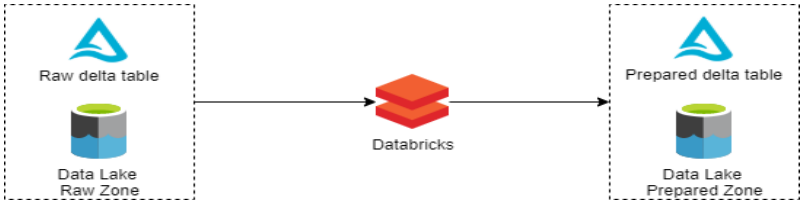

Figure 15. Raw zone to prepared zone data movement layout

The data on the raw zone was extracted using databricks, parsed and basic transformations were done following the client's business requirements only to send the required columns to the prepared area. Streams of data from the raw zone were read and stored in a spark data frame in delta format, using the code below.

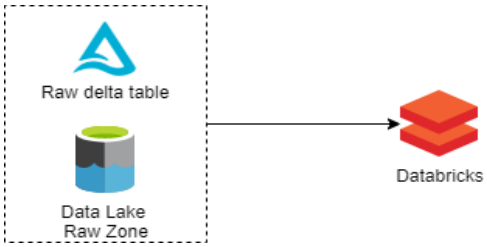

Figure 16. Read data streams from the raw zone

```
dataframe_rawzone = spark.readStream
                        .format("delta")
                        .load(target_table)
```

Figure 17. PySpark read the data stream from raw zone

The components above include:

- **format("delta")**: The file format. The format is delta because it's a delta table.
- **load(target_table)**: contains the location where data was stored. target_table is a predefined variable that included a path to the data found on the raw zone.

Then the delta table column with Avro data was parsed to fetch only the required elements to turn them into columns, making use of the spark SQL functions "col". Example of code is as below.

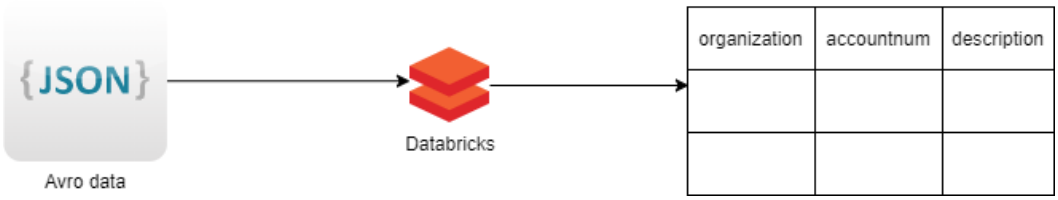

Figure 18. Converting JSON data to columns

```
dataframe_transactions = dataframe_rawzone.select(col("value.afterimage.HX_1").alias("organization"),
                                                  col("value.afterimage.HX_2").alias("accountnum"),
                                                  col("value.afterimage.HX_3").alias("description"),
                                                  col("value.afterimage.HX_4").alias("processdate"),
                                                  col("value.afterimage.HX_5").alias("transactionoccurencedate"),
                                                  col("value.afterimage.HX_6").alias("money"))
```

Figure 19. PySpark parse data from raw zone

Then basic transformations or joins were done to obtain a delta table that was going to reflect the business requirements given by the client.

```
dataframe_transactions.createOrReplaceTempView("TransactionView")

sql_query = '''SELECT  organization, accountnum, description,
                       processdate,transactionoccurencedate,money
               FROM TransactionView tv
               LEFT JOIN  platform.refrencedata plt ON plt.truetransact = tv.accountnum'''

df_transactions_modified = spark.sql(sql_query)
```

Figure 20. Performing transformations on spark dataframe

The components of the above include:

- **createOrReplaceTempView**: creates a view of the spark dataframe.
- **sql_query:** variable containing the SQL query.
- **spark.sql()**: runs the predefined SQL query to obtain a new dataframe stored in a variable. df_transaction_moditified is the variable containing the obtained spark dataframe.

Once the data transformation was over, the spark dataframe was then written to the prepared(silver) delta table on the data lake prepared zone.

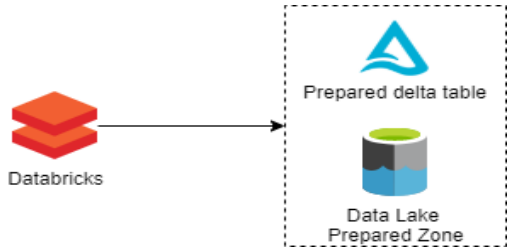

Figure 21. Sending data to the prepared delta table on the prepared data lake zone

```
df_transactions_modified.writeStream.format("delta").\
            outputMode("append").option("checkpointLocation",checkpoint_transaction).\
            start(table_location)
```

Figure 22. Writing data to the prepared zone

The components of the above include:

- **format("delta")**: Format is delta because it's a delta table
- **outputMode**: mode in which data adds to the prepared delta table. Structured streaming runs by default in append mode, but can also run in an update or complete mode.
- **checkpointLocation**: Output sink where the end-to-end fault-tolerance can be guaranteed, developers specify the location where the system will write all the checkpoint information. In the case of stream write failure, the ending location is stored so that once the stream starts, it will

begin from the point it previously ended. The checkpoint location is a directory in an HDFS-compatible fault-tolerant file system. Checkpoint_transaction is a predefined variable which contains the checkpoint path on HDFS.

- **start()**: has the path where data writes send data. table_location is a predefined variable that held the path of the prepared delta table on the prepared data lake zone.

Building the Curated zone

This section focused on performing more complex transformations or aggregations on the data from the data lake prepared zone and sending the data to the curated delta table on the data lake curated zone.

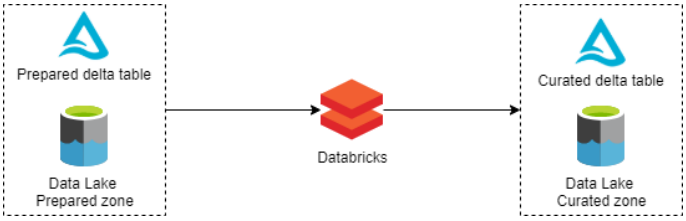

Figure 23. Prepared zone to curated zone data movement lay out

Data streams from the prepared zone delta table were read and stored in a spark dataframe within Databricks.

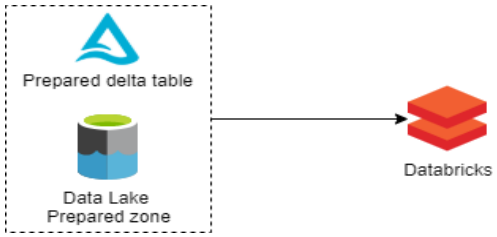

Figure 24. Read data streams from prepared delta table

```
dataframe_preparedzone = spark.readStream
                                .format("delta")
                                .load(target_table)
```

Figure 25. Read data stream from prepared zone

The components above include:

- **format("delta")**: The file format. File formats on delta lakes are delta.
- **load()**: contains the location where data will be gotten; target_table is a predefined variable that includes a path to the data found on the prepared delta table in the data lake prepared zone.

Once the data was available in a spark dataframe, more complex transformations and aggregations were done on the dataframe to produce new columns to satisfy the business requirements given by the client.

```
dataframe_preparedzone.createOrReplaceTempView("TransactionsView")

sql_query = '''SELECT organization, accountnum, description,
                                processdate,transactionoccurencedate,
                                moneytrans,
           CASE WHEN (unix_timestamp(current_timestamp) - unix_timestamp(trans.processdate)) < 31556926
           THEN  (31556926 - (unix_timestamp(current_timestamp) - unix_timestamp(trans.processdate)))
           ELSE 1
           END AS ttl
           from TransactionsView trans '''

dataframe_curated = spark.sql(sql_query)
```

Figure 26. Perform transformations or aggregations on a spark data frame

The components of the above include:

- **createOrReplaceTempView**: creates a view of the spark data frame
- **sql_query**: variable containing SQL query
- **spark.sql()**: runs the predefined SQL query to obtain a new data frame stored in a variable.

Once the final data frame was available, it was then written on the curated(gold) delta table on the data lake curated zone.

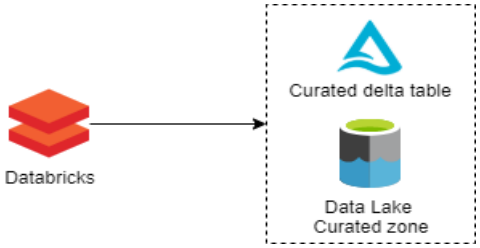

Figure 27. Writing data to the curated delta table on the curated data lake zone

```
dataframe_curated.writeStream.format("delta").\
            outputMode("append").option("checkpointLocation",checkpoint_transaction).\
            start(table_location)
```

Figure 28. Writing data to the curated zone

The components of the above include:

- **format("delta")**: Format is delta because it's a delta table.
- **outputMode**: mode in which data adds to the curated table.
- **checkpointLocation**: Output sink where the end-to-end fault-tolerance can be guaranteed, developers specify the location where the system will write all the checkpoint information. In the case of stream write failure, the ending location is stored so that once the stream starts, it will begin from the point it previously ended. The checkpoint location is a directory in an HDFS-

compatible fault-tolerant file system. Checkpoint_transaction is a predefined variable which contains the checkpoint path on HDFS.

- **start()**: Has the path where data writes send data. table_location is a predefined variable that held the path of the curated delta table on the curated data lake zone.

Sending data to Cosmos DB

In this section, the data present on the curated delta table on the curated data lake zone was extracted using Databricks and sent to Cosmos DB.

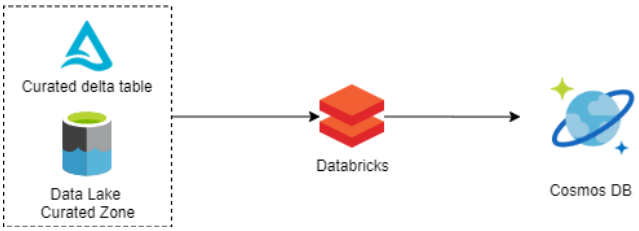

Figure 29. Curated zone to cosmosDB data movement layout

For this part, data streams from the curated zone were obtained and stored within a Spark dataframe.

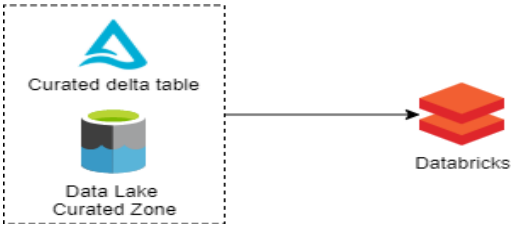

Figure 30. data extraction from the data lake curated zone

```
dataframe_curatedzone = spark.readStream
                                .format("delta")
                                .load(target_table)
```

Figure 31. Read the data stream from the curated zone

The components above include:

- **format("delta")**: The file format. The format is delta because it's a delta table.
- **load()**: contains the location where data was stored. target_table is a predefined variable that includes a path to data found on the curated delta table on the curated data lake zone.

Cosmos DB is the final location of the information processed across the data pipeline, where business applications will consume the data from there to provide it to end-users who are internal or external clients. Additionally, we made Cosmos DB configurations in Databricks before sending data to Cosmos DB. The Cosmos DB where data will be written and its parameters were configured as of below.

```
cosmosdb_write_configuration = {
                        "Endpoint": COSMOS_KEY_URI,
                        "Masterkey": COSMOS_KEY,
                        "Database": "AllTransactions",
                        "Collection": "AllTransactionContainer",
                        "Upsert": "true",
                        "WritingBatchSize": "400"
                              }
```

Figure 32. Cosmos DB configuration

The components of the above configuration include:

- **Endpoint**: this is the URI of the Microsoft Azure Cosmos DB account. COSMOS_KEY_URI is a predefined variable containing the URI.
- **Masterkey**: the master key serves to access the account. COSMOS_KEY is a predefined variable containing the key.
- **Database**: contains the database name.
- **Collection**: contains the collection name**.**
- **Upsert**: indicates inserting data if it does not exist and update if it already exists. True indicates upsert will occur.
- **WritingBatchSize**: number of files with each writes' batch.

After the Cosmos DB configuration step, the data from the spark dataframe was written to Cosmos DB.

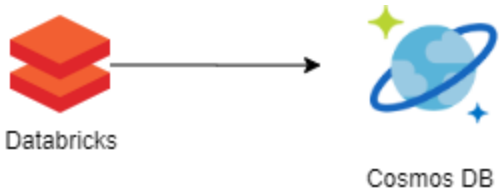

Figure 33. sending data to Cosmos DB

```
dataframe_curatedzone.writeStream
                        .format("com.microsoft.azure.cosmosdb.spark.streaming.CosmosDBSinkProvider")
                        .outputMode("append").option("checkpointLocation", checkpoint_location)
                        .options(**cosmosdb_write_configuration).start()
```

Figure 34. Writing data streams to Cosmos DB

The components of the above include:

- **format (“”)**: indicates the file format.
- **outputMode**: mode in which data adds to cosmos DB database collection. Structured streaming runs by default in append mode, but can also run in an update or complete mode.
- **checkpointLocation**: Output sink where the end-to-end fault-tolerance can be guaranteed, developers specify the location where the system will write all the checkpoint information. In the case of stream write failure, the ending location is stored so that once the stream starts, it will

begin from the point it previously ended. The checkpoint location is a directory in an HDFS-compatible fault-tolerant file system. Checkpoint_location is a predefined variable that contains the path to the checkpoint.

- **Cosmosdb_write_configuration**: cosmos DB configuration defined above.

During this project phase, there were two main challenges. When writing streams of data to delta lake tables, on some occasions, the data from the client's databases was massive (terabytes), hence the file sizes were also large. The file sizes created a problem because in case the spark cluster had few nodes to process the task, it was going to break or become very slow in performing write tasks. The solution identified was to implement "MaxFilePerTrigger" in structured streaming within Databricks. The solution helped to limit the maximum number of files within each micro-batch of data processed. With this solution, even if a spark cluster had few nodes, it could still process data efficiently. The solution was going to be implemented in a later stage of the project towards the end of 2020.

Additionally, on some occasions, late data was coming from the raw zone of the data lake, especially during stream data transformation or aggregations. As a solution, the team decided to implement windowing within structured streaming in Databricks, to ensure that even if stream data were late within data pipelines, it would still be captured within micro-batches of data processed. The solution was going to be implemented in a later stage of the project towards the end of 2020.

4.2 Project 2 - Code Refactoring and Unit Testing

This project focused on refactoring all the ETL pipeline code on Databricks for the MDP project. The code refactored included those written by all Accenture team members. In addition to this, schema and value unit tests were performed on all python functions.

Project Phase 1 - Identifying code smells and refactoring code

Due to NDA agreements the project team signed at Accenture, the detailed reports of SonarQube for the MDP project cannot be disclosed. The screenshots below are examples obtained online that reflect similar steps made by the team during the MDP project.

This first phase focused on identifying code smells and refactoring. Code smells occur when code does not follow fundamental standards of the language in which it is written. Code smells are not bugs or errors; they are absolute violations of the fundamentals of developing software that decrease the quality of code. Refactoring is the activity during which the structure of code is changed to alter its internal structure only and not changing its external behaviour.

Furthermore, to identify this code smells, we used SonarQube, which is an open-source tool created by SonarSource to enable the continuous inspection of code quality to execute automatic reviews with static analysis of code to identify bugs, code smells, and security vulnerabilities. The project team chose SonarQube because it was open source and could easily integrate with Azure workflows.

Moreover, we started by downloading all the code of the MDP project on Databricks to our local computers. Then using SonarQube, we ran code smell analysis to identify bad quality code. To go through the code lines and proceed with correcting the errors, we accessed the code lines in an area within SonarQube similar to the one below, reserved for code modifications.

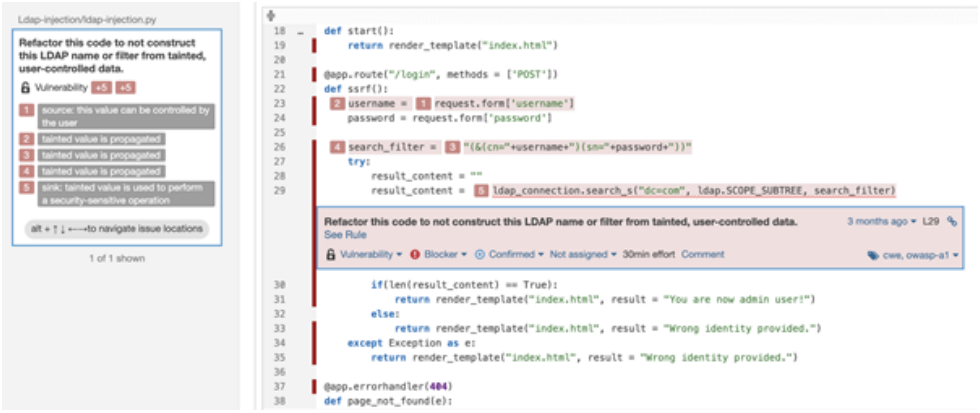

Figure 35. SonarQube code update area

Source: Retrieved from Sonarqube.org

Within this area, we made corrections to the code and refactored following PEP-8 standards, which is a style guide for Python code. After completing all modifications on the code, we then reran the SonarQube analysis to ensure there were no code smells, no bugs vulnerabilities, and zero duplications. After this step, we proceeded to give an update of this project phase during code review sessions.

Project Phase 2 - Unit tests

Once the first phase was completed, we then proceeded to conduct the Unit test in two steps. The first step was to upload the refactored code on Databricks to run the unit tests. Since the MDP project code was in python, the unit test framework within python came up as the best tool to test the project code on Databricks.

Unit testing

A unit test is in-built into the Python standard library. The unit test contains both a testing framework and a test runner. The unit test has essential requirements for writing and executing tests, they include:

- Classes as methods containing all tests.

- Use a series of unique assertion methods in the unit test —use TestCase class instead of the built-in assert statement.

We used the following Unit test general steps:

- Import unit test from the standard library.
- Created a class called, for example, TestName that inherits from the TestCase class.
- Then we converted the test functions into methods by adding 'self' as their first argument.
- Then the assertions were changed to use the self.assertEqual() method on the TestCase class.
- Change the command-line entry point to call unittest.main()

We did the schema and value test on all MDP project code functions. Below is an example of a value test.

```
import unittest
import pandas as pd

class TestFunctionH(unittest.TestCase):

    def runTest(self):

      # 1. ARRANGE

      # pdf_output is built out of this class to serve as  expected output.
      # pdf_output is a pandas dataframe built from a spark dataframe.
      pdf_output

      # 2. ACT

      dataframe_test = functionh(input_dataframe) # apply function to be tested on input dataframe
      pdf_test = dataframe_test.toPandas()        # convert from spark to pandas dataframe

      # 3. ASSERT

      assert pdf_test.equals(pdf_output)          # compare input  dataframe to expected out dataframe


suite = unittest.TestLoader().loadTestsFromTestCase(TestFunctionH)
runner = unittest.TextTestRunner(verbosity=2)
runner.run(suite)
```

Figure 36. Value unit test

The value test aimed at making sure the values of the output data frame from applying the function we were testing on the input data frame was as expected. To ensure that, when we send code to production, the output will always reflect our predefined expected output. Below is an example of the schema test.

```
import unittest
from pyspark.sql.types import *

class TestFunctionHSchema(unittest.TestCase):

    def test_run(self):


      # 1. ARRANGE

      # df_input is a spark dataframe which will serve as input.
      df_input

      # create output schema. This is the expected schema after applying the
      # the function to be tested on the input dataframe
      expected_schema =  StructType([
                                StructField('organization',IntegerType()),
                                StructField('accountnum',LongType()),
                                StructField('description',StringType()),
                                StructField('processdate',StringType()),
                                StructField('transactionoccurencedate',StringType()),
                                StructField('moneytrans',StringType()),
                                ])
      # 2. ACT

      df_test =  functionh(df_input)  # apply function on input dataframe

      # 3. ASSERT

      assert  df_test.schema  == expected_schema   # compare input schema to expected schema


suite = unittest.TestLoader().loadTestsFromTestCase(TestFunctionHSchema)
runner = unittest.TextTestRunner(verbosity=2)
runner.run(suite)
```

Figure 37. Schema unit test

This schema test aimed at making sure the schema of the output data frame from applying the function we were testing on the input data frame was as expected. To ensure that, when we send code to production, the output schema will always reflect our predefined expected output schema.

Once the unit test completed on Databricks, we downloaded all the notebooks to rerun the tests locally this time using PyTest. Note that since the tests were already created on Databricks using the unit test, we just had to re-use those tests to run pyTest locally on computers.

At the end of this, we provided a coverage report to the client's technology team to show the percentage of the MDP project code lines tested.

During this project, since it was the first time we were doing unit test and code refactoring for the MDP, we had to take more time to learn about the PEP-8 standards, Unit test framework best practices and SonarQube. Overall we were able to accomplish the above project in due time.

5. Conclusions

The objective of the MDP project was to migrate the data of the client from their legacy systems to the cloud and build the ETL pipelines on Databricks to process this data across different zones on the data lake.

The MDP project had three development environments, including pre-development, test and production environment. The pre-development environment is the initial environment where developed code got tested to identify loopholes within the MDP project cloud and analytics resources. Once significant checks passed the verification by the project team, and the client's technology and security team approved, all the MDP project code on Databricks got transferred to the test environment which had similar configurations to the production environment.

Furthermore, the production environment is the final area where all MDP project code goes into operation for their final use across the client's infrastructure, to enable the client to run all their daily activities with the new data platform.

All the migration and ETL pipeline code done by the Accenture team passed verification within the pre-development and test development environments. The movement to the production environment was going to occur later on at the end of the year 2020. The work done by the Accenture team handled a large part of the client's infrastructure related to clients (individual and corporate) transactions.

5.1 Connection to the master program

The master in information management with specialisation in knowledge management and business intelligence with its theory and hands-on components empowered the NOVA Information management school student with foundational knowledge in areas of data management, data warehousing, data visualisation, business intelligence, data engineering, and knowledge management. Also, equipped him with foundational knowledge in tools such as python, SQL, Spark, Hadoop, Flume, Databricks, Sqoop, Microsoft SQL Server, and PowerBI.

Likewise, learning about python, SQL, spark, and Databricks before joining the Accenture team helped him to understand faster the project technologies used within the company context and learn more quickly, more complex techniques to manipulate the different tools.

5.2 Internship evaluation

When the NOVA Information management school student joined the applied intelligence team of Accenture to work on the MDP project, the goals of his work were to develop experience in business requirement gathering and translating into a technical requirement. Additionally, make use of big data

and cloud tools such as Azure Databricks to build robust ETL pipelines to extract data, transform and load on cloud data lakes. Also, learn code refactoring and testing.

Likewise, at the end of the student's involvement in the MDP project, all the objectives stated above were attained. He had the chance to work on topics highlighted in the goals above under the guidance of talented professionals who had extensive experience using the technologies and frameworks used in the MDP project.

5.3 Limitations

During the execution of the MDP project, we encountered some challenges detailed below.

The migration of data from legacy systems often took long periods because of new API creation to convert data on the legacy storages to Avro format before pushing to Kafka. This development of new APIs took long periods due to technical challenges, administrative authorisations and getting security clearance validation for the data moved.

Besides, often, decision making within the project to change some tools or business requirements provided by the client due to challenges encountered during the project execution took very long periods. Any change which was going to affect the MDP project had to be validated by the client's technical team and administration in charge of platform development, which took time.

Furthermore, some core components of the project, which were part of the Accenture team's work, went through the client's team for approvals or execution. For example, spark cluster creation and adjustment on Databricks. Still, because the client's team handled this component, there was a constant back, and forth movement in case changes had to be made rapidly because of authorisation approvals, cluster budget restrictions and security clearance which often took much time.

5.4 Lessons Learned

During the MDP project, the NOVA Information Management School student had the opportunity to learn how to use better Databricks, Azure cloud resources such as data lake Gen 2 and Azure Cosmos DB — also test-driven development methodologies by implementing code refactoring and testing.

More so, he learned how to work in diverse teams following the agile methodology and adapting to delivering quality work on time within short sprints.

Bibliography

1. Marr, B. (2015, March 19). Why only one of the 5 Vs of big data really matters. Retrieved October 13, 2020, from https://www.ibmbigdatahub.com/blog/why-only-one-5-vs-big-data-really-matters

2. Acccenture. Our purpose. Retrieved February 22, 2021, from https://www.accenture.com/us-en/about/company-index

3. Strategy & Consulting Services. Retrieved October 13, 2020, from https://www.accenture.com/us-en/about/consulting-index

4. Accenture Interactive. Retrieved October 13, 2020, from https://www.accenture.com/us-en/about/interactive-index

5. Enterprise Technology Services & Solutions. Retrieved October 13, 2020, from https://www.accenture.com/us-en/about/technology-index

6. Business Operations Services & Solutions. Retrieved October 13, 2020, from https://www.accenture.com/us-en/about/operations-index

7. Applied Intelligence Services & Solutions. Retrieved October 13, 2020, from https://www.accenture.com/us-en/services/applied-intelligence-index

8. Data platform. Retrieved October 13, 2020, from https://looker.com/definitions/data-platform

9. What is a data platform?: Emerging Technologies. Retrieved October 13, 2020, from https://www.splunk.com/en_us/data-insider/what-is-a-data-platform.html

10. Radoev, M. (2017). A Comparison between Characteristics of NoSQL Databases and Traditional Databases. *Computer Science and Information Technology, 5*(5), 149-153. doi:10.13189/csit.2017.050501

11. Turban,E.,Sharda,R.,Aronson,J.E.,&King,D.(2008). Business intelligence: A managerialapproach. PearsonPrenticeHallUpperSaddleRiver,NJ.

12. Inmon, W.H.(2005). Building the datawarehouse. JohnWiley&Sons.

13. Steer, A. (2017). The Modern Data Platform: Challenges associated with traditional data warehouses. Retrieved November 26, 2020, from https://itelligencegroup.com/wp-content/usermedia/white-paper-analytics-modern-data-platform-glo-en.pdf

14. http://www.datagovernance.com/adg_data_governance_definition

15. Gartner_Inc. Definition of Scalability - Gartner Information Technology Glossary. Retrieved January 17, 2021, from https://www.gartner.com/en/information-technology/glossary/scalability

16. Mazumder, S., Bhadoria, R. S., & Deka, G. C. (2017). Distributed computing in big data analytics: Concepts, technologies and applications. Cham: Springer.

17. Gartner. Hype cycle for big data, 2012. Technical report (2012) On the role of Distributed Computing in Big Data Analytics 11.

18. Afgan, E., Bangalore, P., Skala, K. Application information services for distributed computing environments. Future Generation Computer System 27 (2011) 173-181

19. Schroeck, M., Shockley, R., Smart, J., Romero-Morales, D., Tufano, P. Analytics: The real world use of big data. Ibm institute for business value – executive report, IBM Institute for Business Value (2012)

20. Cattell, R. Scalable sql and nosql data stores. Technical report (2012)

21. Schroeck, M. et al., 2012. Analytics: The real-world use of big data.

22. Microsoft, 2013. The Big Bang: How the Big Data Explosion Is Changing the World.

23. Manyika, J. et al., 2011. Big data: The next frontier for innovation, competition, and productivity. McKinsey Global Institute.

24. Chen, H., Chiang, R. & Storey, V., 2012. Business Intelligence and Analytics: From Big Data to Big Impact. MIS Quarterly, 36(4), pp.1165–1188.

25. Dumbill, E., 2013. Making Sense of Big Data. Big Data.

26. De Mauro, Andrea & Greco, Marco & Grimaldi, Michele. (2016). A formal definition of Big Data based on its essential features. Library Review. 65. 122-135. 10.1108/LR-06-2015-0061.

27. Gribb, R. What is a Distributed System? Retrieved January 18, 2021, from https://blog.stackpath.com/distributed-system/

28. Gibb, R. (2019, July 26). What is a Distributed System? Retrieved October 26, 2020, from https://blog.stackpath.com/distributed-system/

29. Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. In 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST) (pp. 1–10). IEEE. http://doi.org/10.1109/MSST.2010.5496972

30. KS, B. (2011, April 29). Word Count - Hadoop Map Reduce Example. Retrieved November 23, 2020, from http://kickstarthadoop.blogspot.com/2011/04/word-count-hadoop-map-reduce-example.html

31. Apache Spark. Cluster Mode Overview. Retrieved November 18, 2020, from https://spark.apache.org/docs/latest/cluster-overview.html

32. RabbitMQ. (2020). RabbitMQ – Messaging that just works. Retrieved November 24, 2020, from https://www.rabbitmq.com/

33. Cloudera Inc. Introduction to Kafka. Retrieved November 24, 2020, from https://docs.cloudera.com/HDPDocuments/HDP2/HDP-2.4.2/bk_kafka-user-guide/content/ch_introduction_kafka.html

34. Apache Kafka. (2020). Apache Kafka. Retrieved March 23, 2017, from https://kafka.apache.org/intro

35. R. Kimball and M. Ross, The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. John Wiley & Sons, 2011.

36. B. Ellis, Real-Time Analytics: Techniques to Analyze and Visualize Streaming Data. John Wiley & Sons, 2014.

37. R. Kimball and M. Ross, The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence. Wiley Publishing, 2010

38. A. Sabtu, N.F. M.Azmi, N. N. A. Sjarif, S. A. I., O. M. Yusop, H. Sarkan, S. Chuprat: The Challenges of Extract, Transform and Loading(ETL) System Implementation For Near Real-Time Environment, 2017.

39. T. Jain, R. S, and S. Saluja, "Refreshing Data warehouse in Near RealTime," Int. J. Comput. Appl., vol. 46, no. 18, pp. 24–29, May 2012

40. J. Zuters, "Near Real-Time Data Warehousing with Multi-stage Trickle and Flip," in Perspectives in Business Informatics Research, 2011, pp. 73–82.

41. A. Wibowo, "Problems and available solutions on the stage of Extract, Transform, and Loading in near real-time data warehousing (a literature study)," in 2015 International Seminar on Intelligent Technology and Its Applications (ISITIA), 2015, pp. 345–350

42. Maljic, M. (2020). Delta Lake – extract the real value from Data Lake. Retrieved October 13, 2020, from https://croz.net/news/delta-lake-extract-the-real-value-from-data-lake/

43. Element61. (2020, May 16). Modern Data Platform. Retrieved December 01, 2020, from

https://www.element61.be/en/competence/modern-data-platform

44. Microsoft. What is Azure Databricks? Retrieved December 01, 2020, from https://docs.microsoft.com/en-us/azure/databricks/scenarios/what-is-azure-databricks

45. Gribb, R. What is a Distributed System? Retrieved January 18, 2021, from https://blog.stackpath.com/distributed-system/

46. Favaretto, M., Clercq, E., Schneble, C., & Elger, B. (2020, February 25). What is your definition of Big Data? Researchers' understanding of the phenomenon of the decade. Retrieved October 13, 2020, from https://doi.org/10.1371/journal.pone.0228987

47. Borthakur, D. (2008). HDFS Architecture Guide. Retrieved November 22, 2020, from https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

48. https://delta.io/